Reactive Probabilistic Belief Modeling for Mobile Robots

DISSERTATION

zur Erlangung des akademischen Grades
doctor rerum naturalium
(Dr. rer. nat.)
im Fach Informatik

eingereicht an der
Mathematisch-Naturwissenschaftlichen Fakultät II
Humboldt-Universität zu Berlin

von
Herr Dipl.-Phys. Jan Hoffmann
geboren am 21.12.1973 in Saarbrücken

Präsident der Humboldt-Universität zu Berlin:
Prof. Dr. Dr. h.c. Christoph Markschies

Dekan der Mathematisch-Naturwissenschaftlichen Fakultät II:
Prof. Dr. Wolfgang Coy

Gutachter:

1. Prof. Dr. sc. Hans-Dieter Burkhard
2. Prof. Dr. Uwe Schwiegelshohn
3. Prof. Dr. Raul Rojas

eingereicht am:	1. Juni 2007
Tag der mündlichen Prüfung:	17. Dezember 2007

Abstract

Despite the dramatic advancements in the field of robotics, robots still tend to exhibit erratic behavior when facing unexpected situations, causing them, for example, to run into walls. This is mainly the result of the robot's internal world model no longer being an accurate description of the environment and the robot's localization within the environment. The key challenge explored in this dissertation is the creation of an internal world model for mobile robots that is more robust and accurate in situations where existing approaches exhibit a tendency to fail.

First, means to avoid a major source of localization error – collisions – are investigated. Efficient collision avoidance is achieved by creating a two-layered model of free space in the direct vicinity of the robot. The model is based on camera images and serves as a short term memory, enabling the robot to avoid obstacles that are out of sight. It allows the robot to efficiently circumnavigate obstacles. Furthermore, the model provides information about visual occlusions in the camera image and is thus an important building block of negative information (see below).

The motion model of the robot is enhanced by integrating proprioceptive information. Since the robot lacks sensors dedicated to proprioception, information about the current state and configuration of the robot's body is generated by comparing control commands and actual motion of individual joints. This enables the robot to detect collisions with other robots or obstacles and is used as additional information for modeling locomotion. It allows the robot to reliably move about based solely on dead reckoning as long as it does not detect collisions. If a collision occurs, the belief quickly changes to reflect the uncertainty introduced by this disturbance.

In the context of sensing, the notion of negative information is introduced. Negative information marks the ascertained absence of an expected observation in feature-based localization. This information is not used in previous work on localization because of the several reasons for a sensor to miss a feature, even if the object lies within its sensing range: occlusion of the feature by other objects in the environment, sensor imperfections, erroneous image processing, etc. This information can, however, be put to good use by carefully modeling the sensor. Integrating negative information allows the robot to localize in situations where it cannot do so based on landmark observation alone.

Keywords:
Markov Localization, Probabilistic Robotics, Negative Evidence Modeling, Proprioception

Zusammenfassung

Trotz der Entwicklungen der letzten Jahre kommt es in der Robotik immer noch vor, dass mobile Roboter scheinbar sinnlose Handlungen ausführen und z.B. gegen Wände fahren. Der Grund für dieses Verhalten ist oftmals, dass sich das interne Weltbild des Roboters stark von der tatsächlichen Situation, in der sich der Roboter befindet, unterscheidet. Die darauf basierende Robotersteuerung wählt infolge dieser Diskrepanz scheinbar sinnlose Handlungen aus. Die vorliegende Dissertation geht die Problematik an und zeigt Wege auf, die Weltmodellierung robuster und aussagekräftiger zu machen.

Eine wichtige Ursache von Lokalisierungsfehlern stellen Kollisionen des Roboters mit anderen Robotern oder seiner Umwelt dar. Mit Hilfe eines radialen, zweistufigen Hindernismodells wird der Roboter in die Lage versetzt, Hindernisse zu erkennen, sich ihre Position zu merken (auch wenn er gerade nicht in deren Richtung schaut) und Kollisionen zu vermeiden.

Ferner wird in dieser Arbeit eine Erweiterung der Bewegungsmodellierung beschrieben, die die Bewegung in Mobilitätszustände untergliedert, die jeweils ein eigenes Bewegungsmodell besitzen und die mit Hilfe von Propriozeption unterschieden werden können. Bei der Propriozeption handelt es sich um eine Sinneswahrnehmung, die Informationen über den eigenen Körper eines Lebewesens beinhaltet (z.B. momentane Gelenkstellung und dazugehörige Muskelspannung). Mit Hilfe der Servo-Motoren des Roboters lässt sich eine Art Propriozeption erzielen: der momentan gewünschte, angesteuerte Gelenkwinkel wird mit dem tatsächlich erreichten, im Servo-Motor gemessenen Winkel verglichen. Dieser „Sinn" erlaubt eine bessere Beschreibung der Roboterbewegung und der mit ihr verbundenen Bewegungsunsicherheit.

Zur Verbesserung des Sensormodells wird das bisher wenig untersuchte Konzept der Negativinformation, d.h. das Ausbleiben einer erwarteten Messung, genutzt. Bestehende Lokalisierungsansätze nutzen diese Information nicht, da es – neben einer fehlerhaften Lokalisierung – noch weitere Gründe für ein Ausbleiben einer erwarteten Messung gibt: Verdeckungen des Objekts, fehlerhafte oder verrauschte Messungen oder fehlerhafte Nachverarbeitung der Sensordaten. Eine genaue Modellierung des Sensors ermöglicht es jedoch, Negativinformation nutzbar zu machen. Eine Weltmodellierung, die Negativinformation verarbeiten kann, ermöglicht eine Lokalisierung des Roboters in Situationen, in denen einzig auf Landmarken basierende Ansätze scheitern.

Schlagwörter:
Markov Lokalisierung, Probabilistische Robotik, Negativinformation, Propriozeption

Contents

List of Figures

Chapter 1

Introduction

1.1 Robot Perception and Uncertain Knowledge

Mobile robots face the difficult challenge of operating in environments they can only partially observe. They rely on sensors to create an internal model of the environments they operate in. The robot's decisions and actions are based on this internal model and ultimately on the sequence of sensor readings. For the robot to fulfill its task, knowledge about the environment and itself needs to be represented accurately and comprehensively. Maintaining such a model is challenging for several reasons:

- Sensors have limited sensing range and information from the environment can only be gathered through a 'window into the world' provided by it. Information is thus incomplete and often ambiguous.

  In dynamic environments, it is particularly difficult to estimate where objects are as they may move about unnoticed when the robot is not looking.

- In addition to only monitoring a window of the environment, sensors are of limited accuracy. The sensing error is brought about by the physical process of measuring a certain quantity. Even if this error is small for certain sensors, errors accumulate over time, increasing the uncertainty associated with the robot's model of the world.

- Similarly, the outcome of robot action is probabilistic, e.g., a ball that has been kicked by a robot may, statistically speaking, end up in a range of positions.

The problem of uncertainty was not addressed in early works on robotics, when the notion prevailed that better sensors would eventually become available, creating a highly accurate and unambiguous model of the environment. These systems turned out to be unreliable since it is very difficult to create a model that fulfills such high demands [Schwartz et al., 1986]. This resulted in a paradigm shift towards sensor driven behavior-based robot control. The environment itself serves as the model and uncertainty is accounted for by the designer, who has to create robot behaviors that are robust with respect to sensing errors and that can achieve a task relying solely on the available measurements [Brooks, 1986]. These approaches enable robots to perform tasks in real world scenarios, but they are limited to relatively simple applications. More complex tasks require hybrid architectures, where higher level reasoning and path planning is complemented by lower level, sensor-based behaviors [Arkin, 1998]. However, uncertainty was not explicitly modeled until the advent of probabilistic robotics:

Probabilistic Robotics The field of probabilistic robotics models the belief uncertainty [Thrun et al., 2005]. Using the laws of probability, robot sensing and acting is modeled such that the inherent uncertainty is accounted for. It allows for the incorporation of noisy sensor data and the use of multiple sensors of different type and quality. Probabilistic modeling can represent incomplete, ambiguous knowledge about the environment and the robot's state within the environment. It extends into robot control, where actions are selected according to probable configurations of the environment and probable outcomes of actions, e.g., by taking the action that is best suited for most probable cases.

Probabilistic approaches have proven to be very robust in mobile robot applications. However, the additional real time constraint makes for a challenging problem. Mobile robots have high demands on the efficiency of approaches, since they perform tasks using limited sensing and computational resources. The robot's camera, for example, can only cover a limited field-of-view and is used by the robot to gather both information for localization and for tracking objects relevant to its task. Additionally, autonomous robots are often equipped with limited processing power, which further increases the need for efficient algorithms.

Approximate Probabilistic World Model Model-based control approaches, may they be deterministic or probabilistic, have one common feature: if the model does not accurately describe the environment, robot performance degrades. Although probabilistic modeling is much more robust in this respect, limited sensor data and unforeseen events cause the model to digress from the true state of the environment. The probabilistic framework can completely ac-

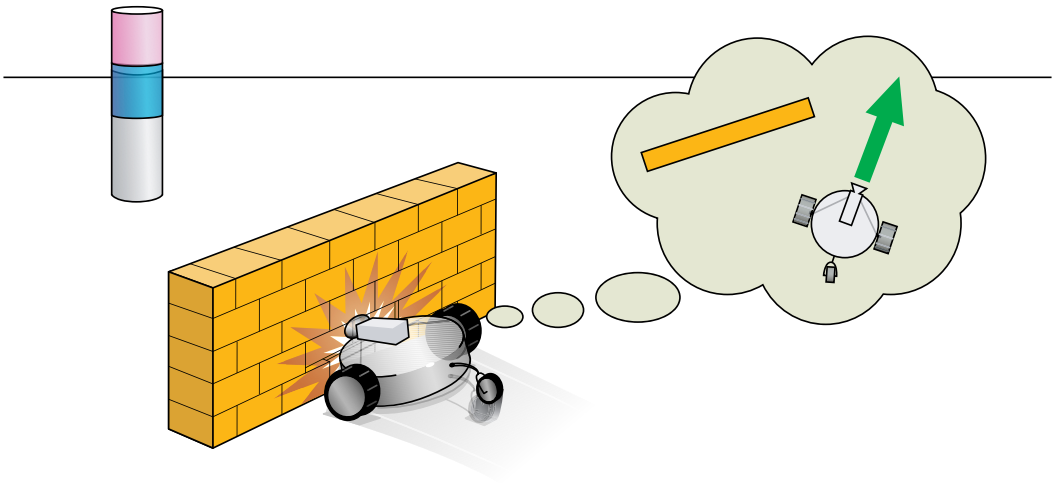


Figure 1.1: Robot accidentally running into a wall, landmark in the background. The robot *believes* it is safe to pass the wall, while in reality it is driving right into it. Future actions are bound to fail due to the mismatch of internal world model and reality.

count for limited sensor data and even unforeseen events, which are modeled by introducing noise as time progresses. In practical implementations in high dimensional state spaces, however, limited computational resources require the use of an approximate, incomplete state representation. Enhancing the commonly used approximate generation of the world model by making better use of the information available to the robot is a key contribution of this thesis.

Robot Running into an Obstacle Let us consider the common problem of a robot running into a wall and its inability to recover from this incident (Fig. 1.1). Such a situation commonly occurs when robot sensors are badly calibrated or environment parameters have changed too much for the robot to adapt. It can be observed in applications ranging from RoboCup to the DARPA Grand Challenge. Even if sensors work as intended, their limited sensing range in conjunction with the potentially dynamic nature of the environment can lead to a disparity between the robot's belief and the environment's true state. After a collision, this disparity increases with time as the robot is unable to gather new evidence (it is staring at the wall, which is not a feature it recognizes) and continues to try to move forward (obviously not resulting in any meaningful movement).

This is a problem specific to model-based architectures, but as sensor-based behaviors heavily rely on adequate observations, they also tend to fail when the robot finds itself in a situation where insufficient sensor data is available.

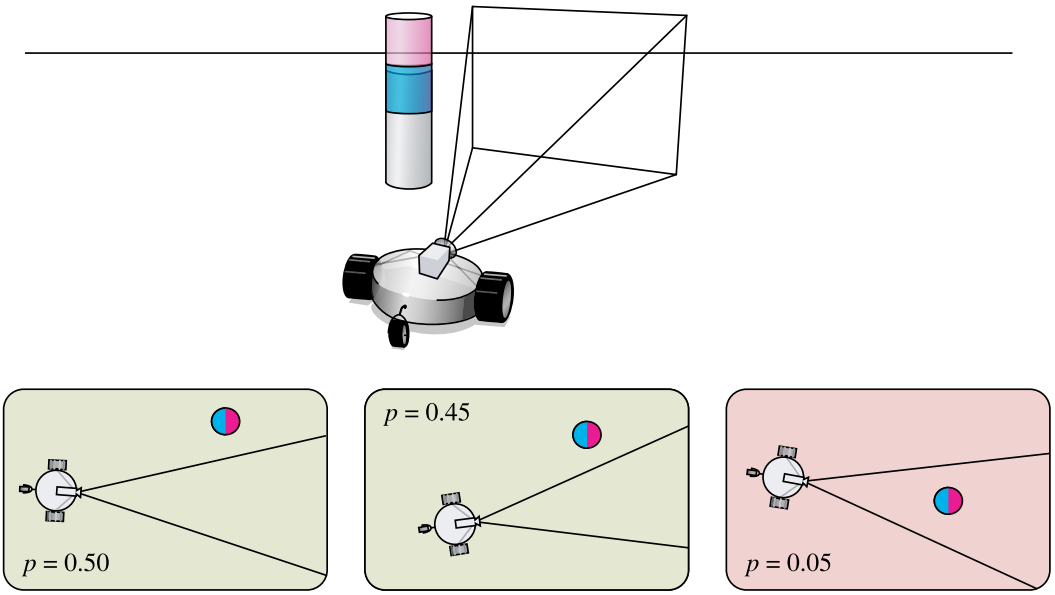


Figure 1.2: Robot not sensing a landmark and three hypothetical robot positions within the environment: two are almost equally likely, the third is unlikely, as it contradicts the current observation. This inference is only possible if the non-detection event is modeled.

Probabilistic robotics offers a framework to model the above described situation. Nevertheless, existing approaches have difficulties coping since they do not model motion and sensing in sufficient detail.

This dissertation investigates approaches to remedy the problem of the robot's world model deviating from the true state of the environment. I will present methods to update this model more frequently and to make it more reactive by using the available information to the fullest.

Proprioception If the robot had some means of detecting that it is bumping into an obstacle, it would be relatively easy for it to recognize the situation and to back up. For robots lacking touch sensors, this information is not readily available. However, sensing the internal state of the body, i.e., proprioception, gives evidence whether or not the robot can move freely and as intended. Collisions result in the robot's actions being hindered, and thus proprioception gives evidence that something is wrong.

Modeling Observations In feature-based localization, features are extracted from the raw sensor data (e.g., landmarks are detected in a camera image). Detected features are used to update the robot's world model. Failure

to detect features, however, is commonly treated as if no new evidence was gathered and the world model is not updated.

A robot that has run into a wall will continue to drive forward having no way of telling that it is stuck, assuming it is continuing its path. However, running into a wall is accompanied by a failure to subsequently detect features. If the robot was able to freely travel forward, it would eventually detect landmarks. Thus, the failure to detect landmarks is evidence that the current localization is not correct and can be used to improve localization (Fig. 1.2).

1.2 Thesis Statement

Document Thesis A mobile robot can achieve a robust, reactive world model under limited sensory and computational resources and can even draw conclusions from the absence of feature detections.

This thesis is substantiated by algorithms, implementations, and empirical evaluation.

1.3 Contributions

Obstacle Avoidance System

The approach to collision avoidance consists of sensing, low-level modeling, plus robot control tailored to the specific characteristics and needs of the application domain. Obstacles are modeled in a two-level representation, which has proven robust and highly efficient for avoiding obstacles. The approach was used successfully in the RoboCup 2003 obstacle avoidance challenge. In RoboCup games, it was used in action selection and to achieve obstacle avoidance. Moreover, it is the basis for modeling occlusions, which is of integral importance when considering negative information.

Proprioceptive Motion Model

This approach equips the robot with a sense of its own body. Since the robot lacks specific introspective sensors, this is achieved by comparing the desired and actual joint angles, allowing the robot to detect if it is able to perform actions as intended. Collisions with other robots and other hindrances can be detected using this method, which then can be used to trigger recovery actions. More importantly, it can be thought of as a sense of proprioception, which can then be used to improve the motion model of the robot. The robot's locomotion and the error associated with it can thus be modeled more accurately, resulting

in an improved, more reactive localization and a better model of the belief uncertainty.

Negative Evidence Modeling

Contrary to existing approaches, the event of not detecting an object can be used as evidence to improve robot localization. This information is mostly disregarded in feature-based modeling approaches, because it is not always easy to determine if an object was truly absent or if the non-detection was caused by other reasons such as occlusions and sensor-imperfections. By using the obstacle model and by carefully modeling the Aibo's camera, the absence of expected sensor readings can be ascertained and discerned from sensing failure. This ascertained non-detection is called *negative information* and it is used to update the current belief. This complementary information allows to continuously update the robot's belief, even when landmarks are not visible.

1.4 Outline

This document starts out with a brief overview of the application domain, RoboCup, and the robot used, the Sony Aibo ERS7 (Chapter 2).

Chapter 3 describes probabilistic modeling of the robot, its environment, and robot-environment interactions. Important laws of probability are recapitulated and the Bayes filter is explained. One of its most successful implementations for localization, the Monte Carlo Localization (MCL), is covered, and details of the implementation used in the experiments are given. Several methods for characterizing uncertainty associated with the belief as well as benchmarking localization are given.

Chapter 4 introduces important sensors used in mobile robots and demonstrates how they are used in the basic navigation task of collision avoidance. A system for obstacle avoidance is presented, which is based on an egocentric, two-layer map of the direct vicinity of the robot. This map allows the robot to safely navigate in its environment, and the overall performance of the system was demonstrated in the RoboCup obstacle avoidance challenge. The obstacle model later plays an important role in ascertaining negative information.

Chapter 5 takes a closer look at the motion model used in the Bayes filter and specifically in the MCL approximation. An approach to incorporate proprioceptive information into the motion model is presented, which better accounts for unsuccessful, unintended outcomes of robot action. This proprioception was first investigated in the context of determining if the robot has run into an obstacle. As the Aibo lacks dedicated sensors to determine if it is touching something, a sense of proprioception was devised based on

comparing control commands and actual motion. This sense is then used to detect collisions and trigger recovery motions. Proprioception is then used to expand the motion model by incorporating information about collisions and other hindrances to generate a more reactive belief representation in the MCL. Experiments show that the resulting belief better models what is actually happening to the robot.

In Chapter 6, robot sensing is examined more closely. First, the sensor model used to incorporate landmark detections into the feature-based localization is derived. The notion of negative information is then presented in two thought experiments and I will argue why it is generally difficult to utilize such information. A sensor model for the robot's camera is derived that specifically includes ascertained non-sensing events. This model is used to integrate negative information into the Bayesian update process. Various experiments show the scope and strength of using negative information.

Chapter 7 concludes the thesis with a summary of the key ideas and explores possible areas of future research.

Publications Most of the presented concepts have been published in refereed conference papers. The obstacle model and avoidance approach was presented in Hoffmann, Jüngel, and Lötzsch [2005] and won the RoboCup 2003 obstacle avoidance challenge. The basis of proprioception for collision detection was presented in Hoffmann and Göhring [2005], its application to modeling the belief was introduced in Hoffmann, Spranger, Göhring, and Jüngel [2006b] and further refined in Hoffmann [2007]. The notion of negative information was introduced and later refined in Hoffmann, Spranger, Göhring, and Jüngel [2006a,b] and Hoffmann, Spranger, Göhring, Jüngel, and Burkhard [2006c].

Chapter 2

RoboCup

2.1 Introduction

In this chapter, a brief description of the robot used and of the application domain for our experiments is given. This is followed by an overview of the participation in RoboCup of the Aibo Team Humboldt and also of the German national robot soccer team, the GermanTeam, a successful collaboration of German universities.

RoboCup is an international research and education initiative. It was first proposed by Kitano et al. in 1995 to advance artificial intelligence and robotics research by providing a standardized test bed for examining novel approaches and technologies [Kitano et al., 1997b,a]. The official mission of RoboCup is to "By the year 2050, develop a team of fully autonomous humanoid robots that can win against the human world soccer champion team." The RoboCup robotic soccer world championship was first held in 1997 in Nagoya, Japan. It was met with great interest in the robotics community and has grown and diversified ever since. Competitions are held in different leagues using various types of robots or software agents: Small-Size League, Middle-Size League, Sony Four-Legged League, Simulation League, and Humanoid League. In the Small- and Middle-Size League, participants are allowed and required to build their own robots within certain size constraints. Teams are largely free to choose actuators and sensors. In the Sony League, the Sony Aibo must be used, i.e., all contestants use the same hardware. In the Simulation League, 11 agents per team are simulated. While the original Simulation League was limited to two dimensions (2D) using a coarse, time discrete model of the environment, currently a transition to the 3D Simulation League is under way featuring, among other advances, a more accurate physical simulation. As the name implies, anthropomorphic robots are used in the Humanoid League. There are numerous competitions within the humanoid league, varying in robot

size and in the specific task.

In addition to the actual soccer tournament, technical challenges are held in the different leagues, trying to raise the bar and push development of technologies that are needed to meet the long term mission goal. Such challenges commonly precede rule changes towards a more realistic, less well-defined environment, such as an increase in the number of players, a decrease in the number of beacons, larger field dimensions, less strict color coding and more natural lighting conditions.

In addition to the above mentioned competitions, two additional branches of RoboCup have developed, RoboCup Rescue, focussing on agents in disaster scenarios, and RoboCup Junior, which is geared towards using robots in high school education.

The research presented here was funded by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG), SPP 1125 “Cooperative Teams of Mobile Robots in Dynamic Environments.”

2.2 Sony Four-Legged League

2.2.1 The Aibo Robot

In contrast to other RoboCup leagues, in the Sony Four-Legged League, all teams use the same platform, the Sony Aibo. The Aibo was conceived as an entertainment robot, but a first demo of the Aibo in the context of RoboCup was encouraged and supported by Sony in 1998 using pre-production prototypes [Fujita and Kitano, 1998]. Currently, the third generation of Aibos is being used in the competition (see Fig. 2.1). The ERS-7 is the successor to the ERS-210 and ERS-110. Over the years, Sony has constantly put effort in improving the Aibo. While using the Aibo makes us as researchers dependent on a commercial product, using the same platform throughout a league has advantages in that is allows for easy sharing and benchmarking of program code.

The open-R software development kit, which in the beginning was not available to the public but only to institutions participating in RoboCup, is used for programming the robot and gives access to the robots sensors and effectors. The robot itself has a 567 MHz MIPS CPU and 64 Mb of RAM. It boots either from internal ROM or from a so called programming memory stick. Driven by servo motors, each leg has three degrees of freedom (DOF), the head has 3 DOF as does the robot’s tail. Contrary to earlier models, the DOFs of the ERS-7’s head are not independent of each other, its the head can be panned in one joint but tilted by using the neck tilt joint and/or the head tilt joint slightly above it.

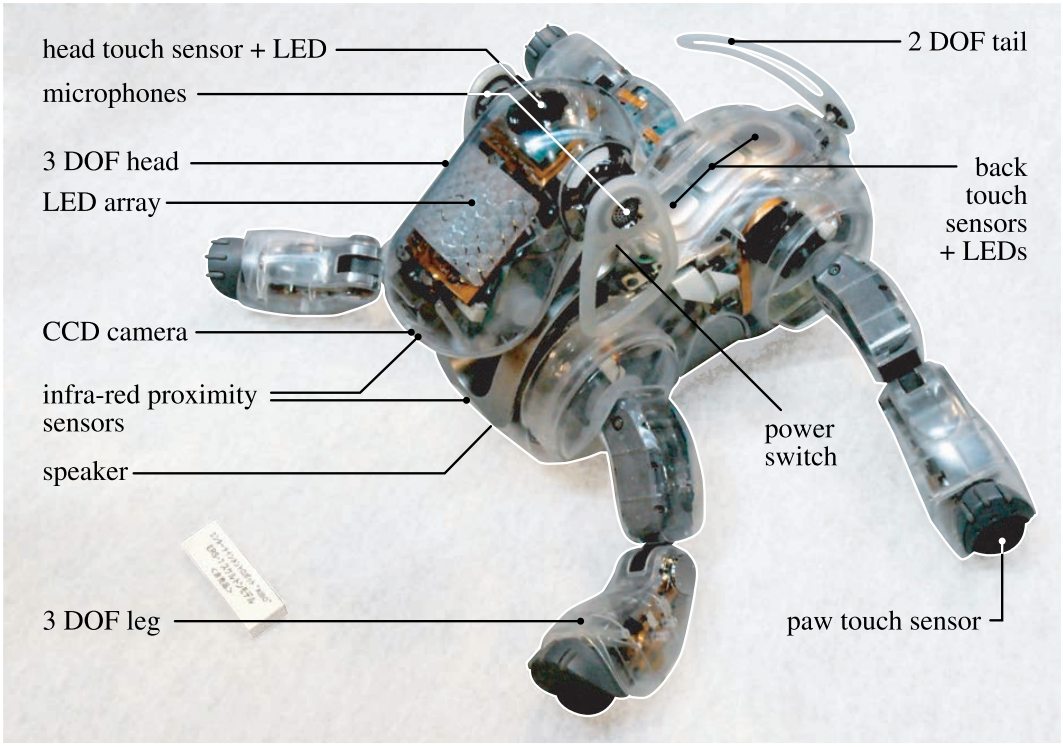


Figure 2.1: Sony Aibo ERS-7, transparent shell to better see the inner workings of the robot. The robots used in the games have an opaque white shell.

The camera is mounted in the robot's head and can thus be panned and tilted. It has a resolution of 208×160 YUV at 8 bits/channel. YUV is a specific color space in which the intensity is stored in the Y channel (luminance) and channels U and V store color information (chrominance).

The robot further features an accelerometer, which is used to detect if the robot has fallen over, 3 infra-red proximity sensors, a thermostat, 2 microphones, touch sensitive switches in the paws, and various buttons and LEDs to allow human-robot interaction. The robot is powered by a battery that allows for a maximum of two hours of autonomous operation.

In January 2006, Sony discontinued the production and sale of the Aibo and also halted all of its robotics projects. The future of the Sony League is therefore uncertain.

2.2.2 Rules of the Game

The games in the Sony League are played 4 on 4, the field is $6\,\text{m} \times 4\,\text{m}$ in size. The game is divided into two halves of 10 minutes length. The robots are fully autonomous. They can communicate among each other using wireless LAN

Figure 2.2: In game situation at the RoboCup 2005 in Osaka, Japan. The field no longer has solid field borders, it is defined only by field lines. Four cylindrical, distinctly color coded beacons help the robot localize. The goals are also uniquely colored.

802.11B. An external computer called the "game controller" is used to send game commands to the robot, such as "kick off," information about penalties, etc.

Soccer Field The environment is colored coded (Fig. 2.2): the ball is orange, players have either blue or red stickers on them depending on the team they play in, the goals are yellow and blue, and there are 4 cylindrical, uniquely color coded beacons just outside the actual field. The exact position and configuration of field lines is defined in the rules. The field lines are white and the carpet is green.

Rule Book The rules are specified after the official FIFA soccer rules, but take into account the specific abilities of the robots. These extensions to the FIFA rules aim at enabling fluid and fair games with as little as possible external human intervention. They are refined every year to make the game more and more life-like. The rule book defines the dimensions and colors of the field, the ball, the markers used on the robots, the beacons, and the number of players and the robot model used. It also defines how standard situations

such as kick-off, penalty kick, and throw in are handled. Certain robot actions are forbidden and are penalized in a game, such as holding the ball for too long, pushing other players, and illegally entering the own penalty area. This *illegal defender* rule states that only the goalie of a team is allowed to be inside the penalty area; Given the size of the robots, it would otherwise be impossible to score a goal. The current rules are available for download from the Four-Legged League home page[1].

2.3 GermanTeam

Due to the strong interest of several German research institutes to participate in RoboCup, establishing a German national team on the common basis of the Sony robot platform was proposed in 2000. The goal of this was to speed up development, encourage collaboration and gain experience in large team, distributed development spanning multiple universities. After being approved by the international committee of the RoboCup organization, the GermanTeam was founded in spring of 2001 with participation of the Humboldt University Berlin (Prof. Burkhard), the Freie Universität Berlin (Prof. Rojas), the Universität Bremen (Prof. Krieg-Brückner), the Technische Universität Darmstadt (Prof. v. Stryk), and the Universität Dortmund (Prof. Schwiegelshohn, Prof. Banzhaff).

The team was established during a kick-off workshop organized by Humboldt University because of our prior experience in RoboCup. At each one of the universities, groups of 10 or more students work on the project. Meetings are held regularly three to four times a year. Communication and documentation is facilitated through a newsgroup, a wiki, and the team report. Along with making the source code used in the tournament publicly available, the team report offers detailed documentation of the code and the concepts behind it. This enables other researchers to work with our code and is also a useful resource for new students entering the project.

The Technische Universität Darmstadt was the first university besides Humboldt University to compete with their own team at the German Open 2001. At the RoboCup 2001 in Seattle, the joint GermanTeam was able to compete for the first time. Following the RoboCup 2001, the software architecture was completely refitted and adapted to the special needs of the GermanTeam. The architecture used is highly modular, facilitating the parallel development of solutions by separate project groups. For example, a new approach to image processing can be developed and tested next to an existing solution without having to change the present, running system. The software

[1] www.tzi.de/4legged

architecture also allows to switch between modules at runtime, allowing objective testing, comparison, and benchmarking. This architecture formed the basis of the German Open 2002 tournament. In the run-up of that year's and also all following German Opens, each of the four participating universities developed special solutions (modules) and competed with a distinct set of modules (e.g., there were four different behavior-control modules). Following the tournament, a thorough analysis of the competition and of the characteristics of the four teams was performed by the GermanTeam, identifying the most successful and promising approaches. A module configuration was then derived and individual modules and their interplay were further optimized and then used in the following world cups. Also, the annual RoboCup GermanOpen tournaments have proven to be an important source of motivation to the team members. The modular software architecture allows each group to focus on their respective main point of interest.

From 2002-2005, the GermanTeam was comprised of the following institutions:

Aibo Team Humboldt Humbold-Universität zu Berlin, Department of Artificial Intelligence, Prof. Hans-Dieter Burkhard, www.aiboteamhumboldt.com

Bremen Byters Universität Bremen, Technologie-Zentrum Informatik, Prof. Krieg-Brückner, www.bremenbyters.de

Darmstadt Dribbling Dackels Technische Universität Darmstadt, Fachgebiet Praktische Informatik/Simulation, Prof. O. v. Stryk, robocup.informatik.tu-darmstadt.de

Microsoft Hellhounds Dortmund Universität Dortmund, Institute for Robot Research (IRF), Prof. U. Schwiegelshohn, www.microsoft-hellhounds.de

At RoboCup 2002 in Fukuoka, Japan, the GermanTeam reached the quarter finals to be defeated only by the two finalist teams. Compared to the strength of other participating teams, the GermanTeam had managed to come close to the leading teams.

The successful cooperation among the GermanTeam members flourished in the following years. Up to now, no other RoboCup participant has managed to establish a national team or similar cooperation that is anywhere near as successful as the GermanTeam.

By 2003, a solid foundation had been established. This not only applies to the software architecture but particularly to individual modules. It enabled

Figure 2.3: Members of the GermanTeam cheering after winning the world championship in the Sony 4-Legged League at RoboCup 2004, Lisbon (Portugal).

us to win the technical challenge at RoboCup 2003 in Padua, Italy, and later helped to smoothly transition to the Aibo ERS7.

When looking at our progression in the RoboCup 2004 tournament in Lisbon, Portugal, it becomes apparent that the GermanTeam not only performed extremely well, it also surpassed many of the other teams that did well in the previous years (Quarter finals: 9:0 victory over the Carneggie Mellon University team; semi finals: 9:2 victory over University of Newcastle, *Nubots*, Australia). Only the opponent in the finals, *UTS Unleashed* (University of Technology Sydney), posed a challenge, but was defeated 5:3. The GermanTeam became world champion for the first time in 2004 (Fig. 2.3). That same year, the Aibo Team Humboldt was able to also win the GermanOpen, underlining its important role within the GermanTeam.

In 2005, the GermanTeam was able to successfully defend its first place at the RoboCup championship in Osaka, Japan.

In 2006, the Microsoft Hellhounds left the GermanTeam to participate as a team of their own. In the Robocup championship in Bremen, Germany, both

teams reached 3rd and 4th places, respectively.

GermanTeam Based Teams In the simulation league, it has been common practice for the teams to release the source code shortly after the tournament. The RoboCup rules allow other teams to participate using such previously released code as a basis, as long as the code has been changed substantially. The league's technical committee decides if the code has been changed 'substantially' based on the team's written application. As the Sony league is similar to the simulation league in essentially being a software league, teams greatly benefit from being able to experiment with and evaluate actual source code of competitors. Prior to RoboCup 2004, we tested our development against a team of robots running the code of the 2003 winners, CMU. In 2004, the Hamburg Dog Bots and the Dutch Aibo Team participated in the German Open for the first time using code based on the GermanTeam 2003 code release, not being members of the GermanTeam.

After the GermanTeam won the 2004 world championship, the code release was met with increased interest, so that in the RoboCup 2005 event, a number of teams participated using the GermanTeam code base. In 2006, this number continued to grow: Hamburg Dog Bots (Universität Hamburg, Germany), Dutch Aibo Team (a national team of 8 universities and institutions), SPQR (Universita di Roma "La Sapienza" and Rome and University of Palermo, Italy), Wright Eagle (University of Science and Technology of China), sharPKUngfu (Peking University, China, participation in 2006), TecRams (Tecnológico de Monterrey, Mexico, 2006). Note that not all of these teams are newcomers to RoboCup. SPQR, for example, had been participating for years before it made the switch to the GT code base.

2.4 Championship Results

2.4.1 RoboCup German Open

In 2001, shortly after the GermanTeam was formed, the first four-legged tournament of the German Open was held. Performance of Aibo Team Humboldt in the German Opens:

German Open 2001

Of the recently formed GermanTeam, only Humboldt University and Technical University Dartmstadt participate; Humboldt 2001 wins, the Darmstadt Dribbling Dackels of the Technical University Darmstadt place second; number of contestants: 4 teams

German Open 2002

Humboldt Heroes places 2nd, the Darmstadt Dribbling Dackels win; number of contestants: 5 teams

German Open 2003

Aibo Team Humboldt places 2nd; Darmstadt Dribbling Dackels win; number of contestants: 8 teams

German Open 2004

Aibo Team Humboldt wins; Darmstadt Dribbling Dackels place 2nd; number of contestants: 10 teams

German Open 2005

Aibo Team Humboldt places 2nd; Microsoft Hellhounds of University of Dortmund, Germany, win; number of contestants: 9 teams

Dutch Open 2006

Aibo Team Humboldt places 2nd; Microsoft Hellhounds of University of Dortmund, Germany, win; number of contestants: 7 teams [2]

2.4.2 RoboCup World Championships

During RoboCup 1998 in Paris, a first demonstration of the Sony Four-Legged League was given using pre-production models of the Sony Aibo. In 1999, the Sony Four-Legged League was established using the ERS-110 model. The following gives a brief overview of the participation and advancement of Humboldt University and the GermanTeam.

RoboCup 1999, Stockholm, Sweden

Humboldt Heroes of Humboldt University participate for the first time; Les Trois Mousquetaires (Versailles Robotics Laboratory, Paris, France) win; number of contestants: 8 teams; Aibo model used ERS-110

[2]Since the RoboCup 2006 was held in Bremen, Germany, the German Open was held in the Netherlands, hence the name change.

RoboCup 2000, Melbourne, Australia

Humboldt Heroes participate; UNSW United (University of New South Wales, Australia) wins, number of contestants: 11 teams

RoboCup 2001, Seattle, USA

The newly formed GermanTeam participates for the first time; UNSW United (University of New South Wales, Australia) wins; number of contestants: 16 teams; Aibo model used: ERS-210

RoboCup 2002, Fukuoka, Japan

GermanTeam reaches quarter-finals, loosing against the later winner of the tournament, rUNSWift (University of New South Wales, Australia); number of contestants: 19 teams

RoboCup 2003, Padua, Italy

GermanTeam wins the technical challenge (see Section 4.4 for details), reaches quarter final only losing against the later finalists, Carnegie Mellon University (USA) and rUNSWift (University of New South Wales, Australia); number of contestants: 23 teams

RoboCup 2004, Lisbon, Portugal

GermanTeam wins championship by winning the final by a score of 5:3 against UTS-Unleashed (University of Technology, Sidney, Australia), number of contestants: 24 teams; Aibo model used: ERS-7

RoboCup 2005, Osaka, Japan

GermanTeam successfully defends the world championship, beating the Nubots (University of Newcastle, Australia) in a penalty shoot-out 4:3 (2:2); places third in the technical challenge, tied with team ARAIBO (The University of Tokyo and Chuo University, Japan); number of contestants: 24 teams

RoboCup 2006, Bremen, Germany

GermanTeam places 4th, Nubots (University of Newcastle, Australia) win; Microsoft Hellhounds place 3rd and win the Technical Challenge; number of contestants: 24 teams

Chapter 3

Probabilistic Modeling

3.1 Uncertainty in Robotics

Robots used in automation and assembly line task do their work in relatively well known, well defined, and well controlled environments. Autonomous mobile robots, on the other hand, have only partial knowledge about their environment, and it is often not well defined and well controlled. In fact, a mobile robot has to infer the state of its often unpredictable and dynamic environment and of itself from limited and noisy sensor data. Such an internal model therefore constitutes an incomplete and ambiguous approximation of the real world.

Early model-based approaches in robot control ignored the problem of uncertainty. The assumption was made that adequate sensing and action mechanisms would eventually be available, providing an accurate model of the environment and allowing deterministic robot-environment interaction (*model-based paradigm*, [Schwartz et al., 1986]). These approaches were limited to application domains where knowledge about the environment is relatively certain and the model is accurate.

The inability to accurately model the robot's environment and the poor performance of robotic systems brought about a paradigm shift in the mid 1980s towards a strong coupling of sensory input to robot action, going as far as to completely reject internal models of the world. In *behavior-based robotics*, actions of *situated agents* (i.e., actual physical robots situated in real environments) are directly controlled using sensor data, explicitly not modeling the environment but instead using the world itself as a model (Brooks' *subsumption architecture*, [Brooks, 1986]). Thus shortcomings in modeling the environment are circumvented by the creation of simple, reactive robot *behaviors*.

While highly successful at simple tasks, behavior-based approaches fail to scale to more complex tasks, which eventually led to *hybrid control* architec-

tures. These architectures use behaviors for low-level control, but also incorporate high-level deliberation and planning to allow the robot to pursue long term goals [Arkin, 1998]. The control architecture employed by the GermanTeam in 2004 was hybrid: a planning level decides what role the robot should take and what behavior should be performed based mainly on global localization. The behavior itself is then executed based on data modeled relative to the robot (although not actual sensor data, e.g., the ball position relative to the robot). 'Good' behaviors work in a variety of situations, implicitly compensating for uncertain sensor information. Sometimes, without necessarily being intended or anticipated by the designer, more complex intelligent behavior emerges from simple behaviors (*emergent behavior* [Steels, 1991]).

The so far outlined approaches have in common that they don't actively take uncertainty into account: model-based robot control requires an accurate model of the environment to function properly, whereas behavior-based robotic control puts all trust in the sensors; both are also unable to model ambiguities.

Probabilistic robotics, however, does take uncertainty into account in modeling the robot's environment and its actions. Probability distributions are used to describe what the robot can infer about the environment from its observations and actions. Robot control is based on probability distribution and can thus better cope with uncertainty and sensor noise. One example that nicely illustrates this is *costal navigation* where a robot clings to walls or other known features of the environment and avoids venturing into open space [Roy et al., 1999]. When comparing two possible paths, the obvious line between starting point and end point may mean that the robot has to travel through areas where there are few features for it to verify its position. The alternative longer path, clinging to a wall, reduces the uncertainty and may, on average, allow the robot to get to the goal more reliably by avoiding getting lost. The straight line solution may be quicker in most runs, but may cause the robot to get lost in some runs, ruining the average performance. Note that choosing a path that reduces uncertainty amounts to active information gathering.

Probabilistic approaches have proven to be highly robust with respect to sensor limitations and noise and work well in dynamic environments. They produce good results even if a limited or simplified model of sensing and action is used. The probabilistic framework is also well suited for combining the sensor data of sensors of potentially different noise characteristics (*sensor fusion*). It is well equipped to to deal with the *kidnapped robot problem*. In this experiment, a robot that was previously well localized is displaced by an un-modeled event (for instance, a person picking it up and carrying it to a different position) and has to recover from this. The difficulty in terms of localization is for the robot to realize that its current localization is no longer valid and needs to be adjusted.

While these properties are obviously very desirable and useful for a robotic system, the implementations are computationally costly and only approximation to the true probability density function describing the robot belief.

Outline In the following section, I will give an overview of the laws of probability that form the basis of probabilistic approaches in robotics. In particular, I will describe the Monte Carlo localization method and the implementation used as a basis for the work that follows.

3.2 Laws of Probability

Quantities of interest such as measurements, controls, states of the robot and its environment are modeled as *random variables*. Instead of describing the quantity by a single value (or a single guess), the quantity is described by a range of values each associated with a probability of occurrence. The probability of the random variable X to take on the value x is written as

$$p(X = x) \tag{3.1}$$

The total probability of all possible values of X is always 1, i.e., there is always an outcome of the experiment. In other words, summing up the probabilities of all possible values of X yields 1:

$$\sum_x p(X = x) = 1 \tag{3.2}$$

In the case of a coin toss, this means that $p(X = \text{tails}) + p(X = \text{heads}) = 1$. For convenience, $p(X = x)$ will be abbreviated $p(x)$.

The above equations describe discrete random variables. If a random variable is continuous, as is typically the case in a sensor measurement (e.g., the weight of some object or an object's speed), instead of summing over discrete possible outcomes, Equation 3.2 becomes an integral over the *probability density function* (PDF) $p(x)$:

$$\int p(x)dx = 1 \tag{3.3}$$

The integral runs over all possible values of the random variable X, e.g., from $-\infty$ to $+\infty$ or a multi-dimensional space. It is important to make the distinction between probability and probability density. A continuous random variable has a probability p of having a range of values $[x_0, x_1]$ based on the probability density $p(x)$.

The probability of the event that the random variable $X = x$ and at the same time the random variable $Y = y$ is called *joint distribution* and given by:

$$p(x, y) \equiv p(X = x \text{ and } Y = y) \tag{3.4}$$

If X and Y are *independent*, the joint probability is just the product of the two probabilities $p(x)$ and $p(y)$.

$$p(x, y) = p(x) \cdot p(y) \tag{3.5}$$

The term *conditional probability* is used if variables are not independent of each other, i.e., when one variable carries some information about the other. It is defined as follows:

$$p(x|y) = \frac{p(x, y)}{p(y)} \tag{3.6}$$

$$p(y|x) = \frac{p(x, y)}{p(x)} \tag{3.7}$$

If X and Y are independent, $p(x|y) = p(x)$, that is, the outcome of one coin toss does not yield information about the next.

From the definition of the conditional probability, the theorem of *total probability* follows:

$$p(x) = \sum_y p(x|y)p(y) \tag{3.8}$$

The definition of conditional probability is also the basis of *Bayes Rule*, which is obtained by substituting Equation 3.7 into 3.6:

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} \tag{3.9}$$

$$\overset{(3.8)}{=} \frac{p(y|x)p(x)}{\sum_{x'} p(y|x')p(x')} \tag{3.10}$$

Bayes rule relates $p(x|y)$ and $p(y|x)$ and is of great importance for probabilistic robotics. In this context, x can be thought of as being the quantity of interest and y the measurement or sensor reading used to estimate x. The probability distribution $p(x)$ describes the distribution before a sensor reading and it is called *prior probability*, whereas a $p(x|y)$ is called *posterior probability*, which is the probability after a sensor measurement y has been incorporated. $p(y|x)$ is called the *generative model* or *sensor model* as it describes how the quantity X brings about a sensor measurements Y. The denominator in Equation 3.9, $p(y)$, does not depend on x. Equation 3.9 is often rewritten using the normalizing constant $\eta = 1/p(y)$:

$$p(x|y) = \eta p(y|x)p(x) \tag{3.11}$$

In actual application (see resampling in Monte Carlo Localization, Section 3.6.2), it is not always necessary to normalize the probability distribution.

Bayes rule can be extended to condition X not only on the random variable Y, but also on a second variable $Z = z$.

$$p(x|y,z) = \frac{p(y|x,z)p(x|z)}{p(y|z)} \tag{3.12}$$

Likewise, Equation 3.5 can be conditioned on z yielding what is called *conditional independence*:

$$p(x,y|z) = p(x|z) \cdot p(y|z) \tag{3.13}$$

As can be seen by rewriting Equation 3.12, this is equivalent to:

$$p(x|z) = p(x|z,y) \tag{3.14}$$
$$p(y|z) = p(y|z,x) \tag{3.15}$$

Equation 3.14 states that if z is known, no additional information is gained by knowing y. This does not, however, imply that the variables x and y are independent of each other:

$$p(x|z) = p(x|z,y) \nRightarrow p(x,y) = p(x) \cdot p(y) \tag{3.16}$$

3.2.1 Normal Distribution

When measuring a certain quantity x, the measurements commonly exhibit a *normal distribution* (Gaussian distribution). When measuring a quantity over and over again, individual measurements x_i will cluster around the true value of x, but there will be a deviation from that value. As the true value x is unknown, it is estimated by taking the mean $\bar{x}$ over all N measurements x_i. The deviation of the measurements from this mean is denoted by the variance σ^2 of the distribution. $\bar{x}$ and σ are known as the first and second statistical moments of the distribution. Probability distributions are characterized by their statistical moments, normal distributions are fully described by first and

second order moments (i.e., all higher moments are zero).

$$x^{(m)} = \frac{1}{N}\sum_{i}^{N}(x_i - \bar{x})^m \tag{3.17}$$

$$\bar{x} = x^{(1)} = \frac{1}{N}\sum_{i}^{N}(x_i) \tag{3.18}$$

$$\sigma^2 = x^{(2)} = \frac{1}{N}\sum_{i}^{N}(x_i - \bar{x})^2 \tag{3.19}$$

Higher order statistics is used to describe and analyze non-gaussian distribution in classification tasks (e.g., principle component analysis). In this work, however, only the first two moments will be considered. The probability distribution of a normal distributed quantity is given by the Gaussian function:

$$p(x) \equiv \mathcal{N}(x, \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right) \tag{3.20}$$

This formula describes a normal distribution of the scalar x. The multidimensional distribution is called *multivariate* and the corresponding normal distribution is characterized as follows:

$$p(\vec{x}) = \frac{1}{\sqrt{2\pi\,\det(\Sigma)}} \exp\Big(-1/2\ (\vec{x}-\vec{\mu})^{\mathrm{T}}\Sigma^{-1}(\vec{x}-\vec{\mu})\Big) \tag{3.21}$$

with the n-dimensional vector of random variables $\vec{x} = (x_1, x_2, \ldots, x_n)$, the mean vector $\vec{\mu}$, and the positive semidefinite, symmetric *covariance matrix* Σ.

The *expectation* of a distribution (which becomes equal to the mean as the number of measurements goes to infinity) is given by:

$$E(X) = \sum_{x} xp(x) \tag{3.22}$$

$$E(X) = \int xp(x)dx \tag{3.23}$$

The covariance of X is calculated using the expectation:

$$\Sigma_X = E(X - E(X))^2 = E(X^2) - E(X)^2 \tag{3.24}$$

3.3 Situated Agent

An agent is called situated if it operates in a physical environment by means of perception and action. Perception (or sensing) gathers information about

the environment using the robot's sensors, which is then used to update the robot's internal model of the environment. Given the internal model, a control system selects appropriate robot actions, such as moving to a different location, manipu-lating part of the environment, or picking up an object.

3.3.1 State

The environment is described by a collection of variables or quantities necessary for the robot to perform its tasks. This is called the state; some typical state variables are:

Robot pose The robot pose describes the location and orientation of the robot in its environment. In 3-dimensional space, the robot's position is described by the three Cartesian coordinates (x, y, z) and its orientation by the three Euler angles (pitch ρ, roll φ, yawθ). In a typical RoboCup environment, the robot moves in the x-y-plane and therefore can be described by (x, y, θ), the other variables are constant.

Kinematic state The kinematic state describes the current actuator configuration of the robot, e.g., the current joint angles of a legged robot or the relative direction of a pan-tilt-camera mounted on the robot. In analogy to biology, state variables describing the internal state of the robot are sometimes called *proprioceptive.*

Environment state The robot's environment is often described by a map, i.e., the location of static objects such as walls, trees, etc. In many robotic applications, landmarks are used for characterizing the environment. Landmarks are objects that stand out in the environment and can be detected reliably. The environment state can further be made up of information about non-static objects such as doors and movable or moving objects within the environment (other robots, humans, the soccer ball, etc.).

Further state variables may be derived from any of the robot's sensors (e.g., temperature, buttons pressed, etc.). The collection of all variables of interest is called *world state.* This is similar to the philosophical concept of the Laplace's Demon, an entity that knows the current state of every single atom in the universe and that can, using the laws of physics, predict the future. A state is said to be *complete* if it is the best predictor of the future. This usually requires a large number of state variables, but certainly does not describe the environment down to atomic level. The definition of the completeness of a state further implies that this snapshot of the environment at time t contains all information necessary to predict the future and that additional knowledge about

prior states, measurements, and actions does not yield additional information. Such a temporal process is known as a *Markov chain*, i.e., variables prior to t do not influence the future unless they are explicitly mediated through x_t. This is an example of conditional independence: if x_{t-1} is complete, knowing all previous actions and sensor measurements yields no additional information. State x_t can be generated iteratively from the initial state x_0, followed by all robot actions and all measurements taken until t.

The complete state is usually not accessible due to sensor limitations, limited processing power, etc. Instead, a subset of state variables sufficient for the given task is chosen called the *incomplete state.*

3.3.2 Robot-Environment-Interaction

The robot interacts with its environment through action and perception:

Perception z_t In the scope of this work, perception is defined as the process of observing or measuring a state variable. The raw measurement data is being processed, e.g., by feature detection, yielding a *percept* (Fig. 3.5). Typical types of raw data are camera images or range scans, typical percepts are the bearing or distance to detected objects or features of the environment. Observations yield information about the environment and the robot's whereabouts within it and thus result in a decrease in belief uncertainty (information gain ≥ 0).

Sensors are modeled by the conditional probability $p(z|x)$, i.e., the probability of making observation z given the robot is in state x. Such modeling describes the statistical nature of sensor readings and sensor noise. It may also be seen as a noisy prediction of the world state.

Action u_t Control actions change the state of the environment. Such actions include robot locomotion and the robot manipulating objects. *Control data* is information about the expected change of the environment after the action, e.g., the expected location and speed of the ball after it has been kicked by the robot. It also includes location changes of the robot: if the robot speed is set to v, its location changes accordingly over time.

In wheeled robots, an odometer is used to monitor the actual turning of the wheels to determine more accurately the distance traveled. In reminiscence of this, data describing the robot's locomotion is commonly referred to as *odometry*, regardless of the actual method of locomotion.

Stochastic Outcome of Actions Actions are usually accompanied by a loss of knowledge since the outcome of actions is always stochastic. Wheels

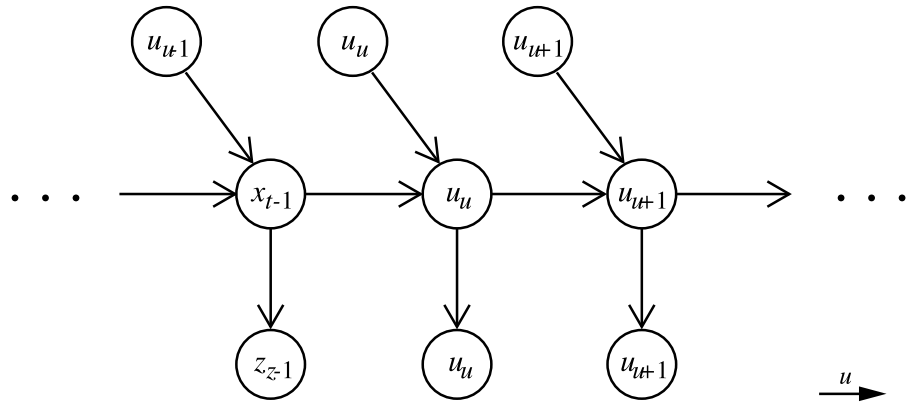


Figure 3.1: Hidden Markov Model of the evolution of the state x over time. The current state x_t only depends on the most recent control action u_t. The state x_t brings about sensor measurement z_t. The state x_t itself cannot be sensed directly, hence the term "hidden."

may be slipping or the robot may not quite hit the ball in its center when kicking it. This is modeled by the *state transition probability*, which describes how the world state changes over time as the robot performs actions u_t:

$$p(x_t|x_{t-1}, u_t) \tag{3.25}$$

Hidden Markov Model The state transition probability and the measurement probability describe the dynamic stochastic system of the robot and its environment as illustrated in Figure 3.1. State x_t depends (stochastically) on the most recent state x_{t-1} and the action u_t. The state x_t gives rise to measurement z_t, in other words: measurement z_t depends on the state at time t. Such a temporal generative model is known as Hidden Markov Model (HMM), since the state is not directly accessible but is instead inferred by means of observation and control.

3.3.3 Belief

The *belief* is the robot's internal model of its environment. It represents the state the robot believes itself and the environment to be in. This may or may not resemble the actual world state: the robot may not have been able to gather enough information just yet, the belief may be uncertain, and it may be ambiguous. The belief is therefore represented by a probability density distribution. The belief distribution assigns a probability to all possible world states. Belief distributions are posterior probabilities over state variables, conditioned on all observations and actions taken:

$$\text{bel}(x_t) = p(x_t|z_0, \ldots, z_t, u_0, \ldots, u_t) \tag{3.26}$$

As we will see in the next section, it is useful to define the belief before incorporating the current, most recent observation after the control u_t. This is called the *prediction* (also: prior probability, prior belief):

$$\text{bel}^-(x_t) = p(x_t|z_0, \ldots, z_{t-1}, u_0, \ldots, u_t) \tag{3.27}$$

Calculating $\text{bel}(x_t)$ from $\text{bel}^-(x_t)$ by incorporating observation z_t is called *measurement update* or *correction*.

3.4 Bayes Filter

The Bayes filter calculates the current belief from the previous state by first incorporating the robot action (yielding the "prior") and then conditioning the prior on the current observation. The state x_t is assumed to be complete, i.e., knowledge about actions and measurements before time t yields no additional information for predicting x_{t+1}. This follows from the *Markov assumption*, which states that past and future data are independent if the current state x_t is known and also complete. Using the Markov assumption, Equations 3.26 and 3.27 become:

$$p(x_t|z_0, \ldots, z_t, u_0, \ldots, u_t) = p(x_t|z_t, u_t) \tag{3.28}$$

$$p(x_t|z_0, \ldots, z_{t-1}, u_0, \ldots, u_t) = p(x_t|z_{t-1}, u_t) \tag{3.29}$$

The *update rule* of the Bayes filter describes how observation and control are used to update the belief:

$$\text{bel}^-(x_t) \longleftarrow \int p(x_t|x_{t-1}, u_{t-1})\text{bel}(x_{t-1})dx_{t-1} \tag{3.30}$$

$$\text{bel}(x_t) \longleftarrow \eta p(z_t|x_t)\text{bel}^-(x_t) \tag{3.31}$$

Starting with the initial belief $\text{bel}(x_0)$ at time $t = 0$, the prior distribution $\text{bel}^-(x_1)$ is calculated by incorporating the robot action creating the prediction (Eqn. 3.30). In mobile robot localization, $p(x_t|x_{t-1}, u_{t-1})$ represents the motion model and describes robot locomotion. Using the prediction (prior belief), the most current measurement is incorporated yielding the posterior probability distribution $\text{bel}(x_1)$ (measurement update, Eqn. 3.31). $p(z_t|x_t)$ is called the sensor model and describes the likelihood of observation z_t given the robot state x_t. After incorporating the sensor information, the process starts

over for the next time step, the filter is executed recursively. A mathematical derivation of the Bayes filter is given in [Thrun et al., 2005].

The initial belief bel(x_0) may or may not contain some knowledge. For instance, it may be a high probability over a small confined area of the state space and zero probability otherwise if the starting point is known, or it may be an uniform distribution over all possible states.

Note that the update rule requires integration over x_t. As x_t may be continuous and a closed form description may not be available, integration is often difficult. Implementations of the Bayes filters therefore are required to somehow approximate the true belief (see below).

As mentioned earlier, the complete state is usually not accessible and the incomplete state is used instead. This may pose a violation of the Markov assumption as the model may not be the best predictor of the future. Other violations are caused by un-modeled system dynamics, inaccurate generative models (for actions and measurements), and approximation errors (e.g., using Gaussian filters vs. Monte Carlo particle filter with a small number of particles). Luckily, Bayes filters have proven to be robust against many of these violations.

3.4.1 Localization

I will use the Bayes filter for mobile robot localization. In an experimental comparison of localization approaches based on the Bayes filter, Gutmann et al. [Gutmann et al., 1998; Gutmann and Fox, 2002] make the distinction between behavior based approaches, those that used dense sensor matching, and those based on using landmarks. The first uses behaviors based on sensing the environment for navigation [Brooks, 1986; Braitenberg, 1984], e.g., following the right-hand rule to move about in an environment and finding back by reversing the procedure [Connell, 1990]. Dense sensor matching uses unprocessed sensor data such as produced by laser range finders to localize the robot in its environment [Dellaert et al., 1999a; Buhmann et al., 1995; Menegatti et al., 2005; Weigel et al., 2002]. Camera images can also be used but require preprocessing in order to make the problem computationally tractable [Dellaert et al., 1999a; Hanek et al., 2002, 2003]. In contrast, landmark- or feature-based approaches first extract features from the sensor data and localization is based on these, e.g., sonar landmarks or the beacons used in RoboCup [Gutmann and Fox, 2002; Röfer and Jüngel, 2003].

Dense sensor matching uses all of the available (raw) data without abstracting from the data and without the need to decide what constitutes a landmark. While it is clearly desirable to use the information to the fullest, the computational cost is often prohibitive, especially in the case of camera

images. In the Sony League, the limited processing power offered by the Aibo and the well defined environment commends landmarks-based navigation.

Another important distinction needs to be made when describing the localization task: whether the localization task is global or local. Global localization has the robot starting out in its environment without any prior knowledge of where it could be. In terms of belief, this is represented as a uniform probability density function over all possible states. In local localization, on the other hand, the robot has some information of where it could be in its environment, greatly simplifying the problem by ruling out ambiguities. Not all of the following implementations of the Bayes Filter can cope with the global localization problem.

3.4.2 Variants and Implementations

As stated before, implementations of the Bayes filter are approximations to the true belief. I here give a brief overview of commonly used algorithms. They are called unimodal if they can model a single hypothesis and the uncertainty associated with it, and multimodal if they can model ambiguities and thus multiple hypothesis. A detailed overview of the approaches including mathematical derivations can be found in [Thrun et al., 2005].

Kalman Filter

The Kalman filter, named after its inventor [Kalman, 1960], constitutes a technique for filtering and predicting linear Gaussian systems. It models the belief as a multivariate normal distribution, which is described by its mean and covariance. The belief is updated during a time step by incorporating the control, yielding the prediction. The control is also modeled as a normal distribution, as is the observation. In the correction step, the observation from the current time step is used to refine the prediction to arrive at a new estimate.

Kalman filter has countless applications in estimation and control and is often used in object tracking. For example, the GermanTeam uses it to track the ball and determine its speed. It offers the greatest accuracy of the filters presented here (if applied to a linear Gaussian system). It is highly efficient when modeling unimodal belief distribution, but, by definition, cannot handle multimodal distributions. An extension that is able to somewhat cope with this is called the multi-hypothesis Kalman filter, which represents the belief as a mixture of Gaussians. The other drawback of the Kalman filter is that it is limited to linear processes. The Extended Kalman Filter and the Unscented Kalman Filter address this limitation and allow non-linear processes to be modeled.

Histogram Filter

The histogram filter decomposes the state space into finite regions or grid cells. A probability is associated with each cell. Regions do not have to be of equal size; the size can represent the significance of a particular region to the task the robot is trying to achieve. Such a representation is called topological; for a robot operating in an office environment, regions could be the individual rooms and distinct positions in the hallway (junctions, corners, doors, etc.). Discretizing the state space in such a way allows the calculation of 3.30 and 3.31, which become sums over grid cells. Histogram filters also work for non-linear processes and allow tracking of multiple hypothesis (as many as there are grid cells). While being more flexible, they become computationally costly for more fine grained decompositions of the state space.

Particle Filter

Particle filters constitute a second type of nonparametric implementations of the Bayes filter. In contrast to histogram filters, the belief is modeled by the density of particles. Each such particle represents a localization hypothesis, generally a robot pose with continuous state variables.

Particle filters allow modeling of non-linear processes and of multiple hypothesis and have been highly successful because of their robustness [Gutmann et al., 1998; Gutmann and Fox, 2002]. I will examine the Monte Carlo particle filter in more detail in the next section, which will be followed by a description of the actual implementation used in the later experiments.

3.5 Monte Carlo Localization

Monte Carlo Localization (MCL) represents the belief by a particle distribution in state space. The belief likelihood is represented by the particle density. MCL is a very popular approach and has been utilized in numerous robot navigation tasks, to name a few, in office environments [Fox et al., 1999a; Dellaert et al., 1999b], in the museum tour guide Minerva [Thrun et al., 2000; Fox et al., 1999b], in the highly dynamic RoboCup environment [Lenser et al., 2001], and outdoor applications in less structured environments [Montemerlo and Thrun, 2003].

Figure 3.2(1) illustrates how the belief is represented by a particle distribution (sample set S consisting of $N = 500$ particles $s^{[n]}$). In the top row, gray scales indicate probability density, in the bottom row, dots represent samples describing the robot position (robot orientation is not shown). If the true belief is known, the particle distribution can be sampled from it. The experiment

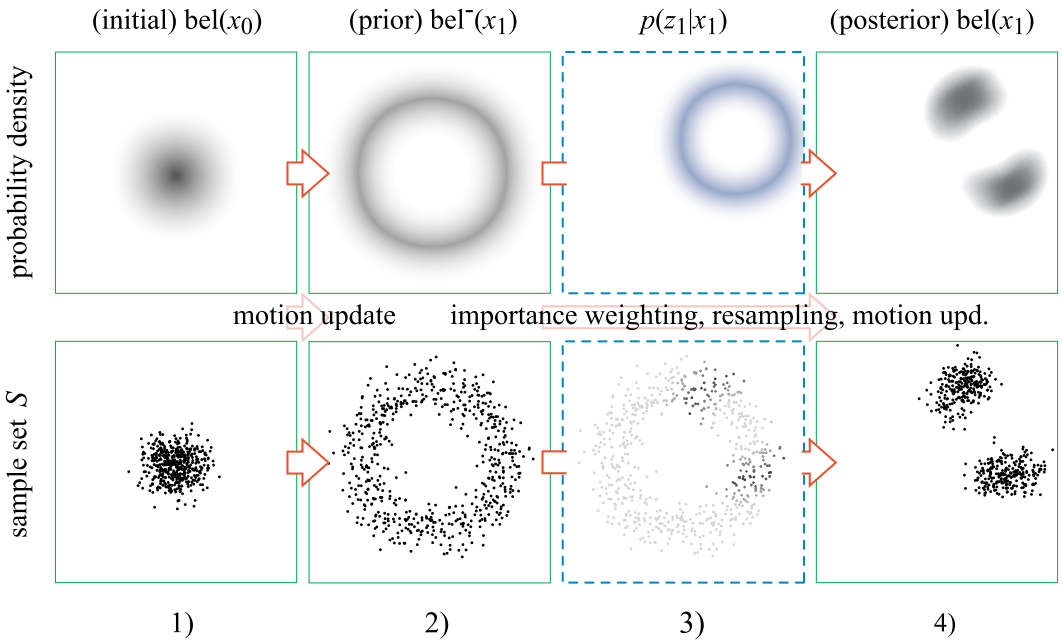


Figure 3.2: Robot belief propagation (top) and its approximation using a particle filter (500 particles, bottom). The robot starts off relatively well localized but at unknown orientation in 1) and moves forward, resulting in the belief shown in 2). It then detects a landmark and measures the distance to it. The corresponding sensor model $p(z_t|x_t)$ is shown in column 3) top. The sensor model is used to weight the particles according to how well they agree to the current measurement, indicated by the intensity in 3) bottom. Particles of high weight are used to generate the new particle distribution shown in 4) in a process called *resampling*. Illustration after [Dellaert et al., 1999b].

starts out with a robot localization that is Gaussian. The corresponding initial belief is shown in the first column of Fig. 3.2.

The belief is propagated using the *motion model*, the probabilistic model describing robot locomotion. Motion is modeled probabilistically as robot actions do not always lead to the intended, deterministic result in the environment. The robot starting position is relatively certain but its orientation is unknown. The current robot pose is estimated only from distance traveled and without external information (perception); this process is called *dead reckoning*. As the robot moves forward, the belief changes according to the motion model (Fig. 3.2(2)). This constitutes the prior belief, which is used as a basis to incorporate the now following sensor reading.

The robot senses a landmark at distance d. For a robot sensor, the probability of sensing the landmark at distance d is given by the probability density

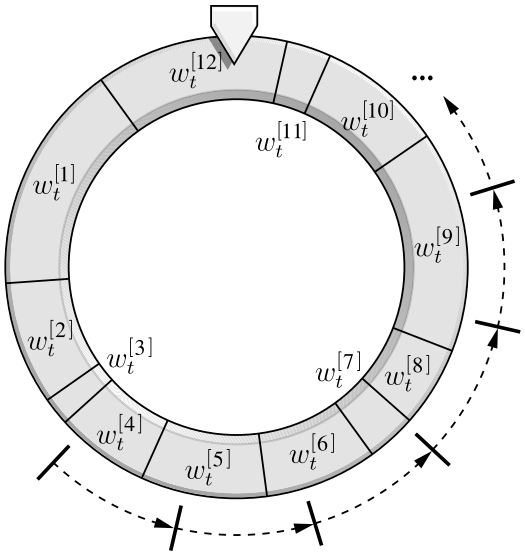


Figure 3.3: Resampling: the particles are arranged on the "roulette wheel" with the width of segments representing the particle weight. Samples for resampling are chosen by repeated turning of the wheel or by first choosing a random starting point and then moving along in equally sized steps. In both cases, particles with higher weights are chosen more frequently to create new particles.

function over the state space $p(z_{\text{LM}} = d|x)$, where z_{LM} denotes the landmark observation. This PDF is called the *sensor model* and is shown in Fig. 3.2(3) top. The sensor model is used to weight the particles representing the belief in accordance to how well they agree to the current sensor reading, i.e., $w_t^{[n]} \propto p(z_{\text{LM}} = d|x^{[n]})$, where x_n denotes the position of particle n. The particle weight is indicated by the shade of gray of particles in Fig. 3.2(3) bottom.

Now that the particles are weighted, the set is used to generate a new set based on the weights of particles. This process is called resampling (see Fig. 3.3). Particles of high weight are more likely to be chosen to generate new particles than those of low weight. This newly generated particle set will have particles representing the same robot position. After the next probabilistic motion update, though, the small errors associated with robot motion will cause the particles to diverge (Fig. 3.2(4)).

After the new sample set has been generated, the algorithm starts over. Figure 3.4 shows the iterative MCL algorithm in pseudocode.

In addition to weighting particles using the observation, particles are sometimes also sampled from the sensor model PDF. This is particularly helpful if a small number of particles is used, during global localization, and in the case

1: **MCL**(S_{t-1}, u_t, z_t)
   // motion update and weighting of particles according to sensor model:
2: $S_t^- = \emptyset$
3: **for** $(n = 1$ **to** $M)$ **do**
4: $x_t^{[n]} =$ **motion-update**$(u_t, x_{t-1}^{[n]})$
5: $w_t^{[n]} =$ **sensor-model**$(z_t, x_t^{[n],m})$
6: $S_t^- = S_t^- + \langle x_t^{[n]}, w_t^{[n]} \rangle$
7: **end for**
   // sample from S_t^- using the weights to generate posterior:
8: $S_t = \emptyset$
9: **for** $(n = 1$ **to** $N)$ **do**
10: draw i with probability $\propto w_t^{[n]}$
11: $S_t = S_t + \langle x_t^{[i]}, M^{-1} \rangle$
12: **end for**
13: **return** $bel(s_t))$

Figure 3.4: Monte Carlo Localization (MCL) algorithm after [Thrun et al., 2005].

of localization failure. In these cases, region of high probability of the sensor model may only be populated by few particles prior to the measurement and convergence can be slow. Generating new particles based on the current observation speeds up convergence; this method is called *sensor resetting localization* [Lenser and Veloso, 2000].

An approach to modeling unforeseen events is to inject random particles. The number of samples of the current sample to be discarded and replaced by random particles is governed by monitoring the current average weight of all particles. This is used as a measure of how well the robot is currently localized and, in the case of kidnapping, can trigger the generation of random particles.

As long as the particle count is sufficiently high, MCL is able to approximate most PDFs encountered in robotics applications, especially complex multimodal, non-Gaussian distribution. Because it uses particles, MCL is good at modeling where the robot might be, but not very good at modeling a largely uniform distribution (this requires a large number of particles). The number of particles determines the accuracy and reactiveness of the filter. As the particle count determines also the runtime $O(M)$, accuracy generally needs to be traded off against computational speed.

The probabilistic nature of the motion model and the sensor model will be investigated in more detail in Section 5.1.1 and 6.1.1, respectively.

3.6 Reference Implementation: GermanTeam Monte Carlo Localization

Early approaches employed by the Aibo Team Humboldt used case based reasoning for localization [Wendler et al., 2001] based on vision data. This approach was abandoned by the GermanTeam in favor of the more robust Monte Carlo Localization. The approach features some noteworthy deviations from the version just described. Many implementation details of the particle filter are due to the limited particle number used. Before the MCL implementation is described in detail, let us first take a closer look at the vision system providing the sensor input for localization.

3.6.1 Vision

Since the robot operates in a color coded environment, camera images are commonly segmented using a color table [Bruce et al., 2000]. Color segmentation maps individual pixels to pre-defined color classes. Such classes are stored in a color table, assigning a set of YUV-values to a certain color class. Such pixel based classification often has troubles dealing with highlights and shadows. To deal with such, information from neighboring pixels can be used [Hanek et al., 2002]. Color tables are created before a run, which means they cannot adjust to changes in the global lighting and will produce bad results if lighting conditions change between calibration and actual run. Automating the task of creating color tables and creating adaptive systems that can robustly classify colors has been the focus of numerous recent publications [Jüngel et al., 2004; Schulz and Fox, 2005; Hanek et al., 2003; Gunnarsson et al., 2006; Simon et al., 2005]. To underline its importance, a "variable lighting challenge" was conducted as part of the RoboCup Technical Challenge in both 2004 and 2005.

From the camera image, features representing objects of interest (i.e., beacons, goals, the ball, other players, field lines) are extracted. Objects close to the robot appear large in the image, whereas far away objects are smaller and near the image horizon. (Given the height and the current pan and tilt of the camera, the horizon can be calculated as the intersection of the plane parallel to the ground on the height of the robot's camera and the projection plane.) To avoid having to process every single pixel in the image, the image is only scanned in a grid that takes into account the possible sizes of objects. The area near the horizon in the image is scanned at high resolution whereas a wide spaced grid is used elsewhere (Fig. 3.5). The grid lines are perpendicular to the horizon. These lines are scanned from the bottom to the top. Field lines are directly detected on the scan lines. Other objects of interest are detected by specialized object detectors ("specialists"), which are triggered if a specific

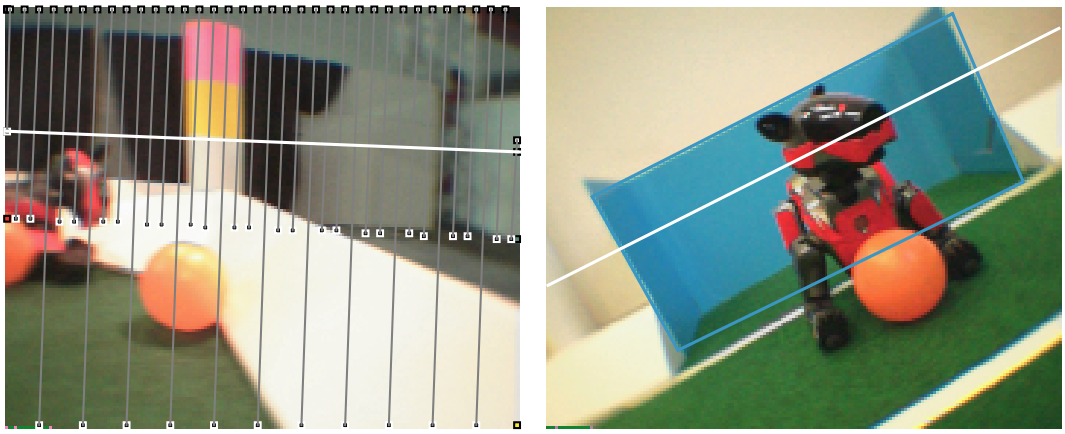


Figure 3.5: Sony Aibo ERS210 camera images. *Left:* Grid used to scan the image. *Right:* Goal percept.

color signature is detected on a scan line. These specialists may leave the grid to determine the exact size of objects in the image. The following objects are detected by specialists: ball, goals, landmarks, and other players. Besides field lines, the line of the goal touching the ground is also detected directly on the scan line. Since the scan starts at the bottom of the image, the free space around the robot can easily be detected (see Chapter 4 for details).

The information found in the image is stored in *percepts*; image processing may yield the following percepts: a ball percept, flag percepts, goal percepts, line percepts, obstacle percepts, and player percepts. Of these, detected flags, goals, and lines are used for localization.

3.6.2 Monte Carlo Particle Filter

The MCL implementation used by the GermanTeam largely works as described above. Due to the limited computational power of the Aibo, the particle filter uses only 100 particles. Each particle represents a robot pose $\vec{r} = (x, y, \theta)$. Furthermore, a weight $w^{[n]}$ is associated with each particle n.

Motion Model

The particles are propagated using the motion model describing the locomotion of the four-legged Aibo. It will be described in detail in Section 5.1.1.

Sensor Model

The percepts generated by image processing and being used for localization fall into two categories: unique landmarks and line percepts. The bearing to

landmarks is used for localization. Since they can be uniquely identified, this constitutes a significant information gain. Particles are weighted using the sensor model, which is approximated by the similarity function describing how well expected bearing ω_x matches the measurement ω_{meas}:

$$p(z|x,m) \propto \text{sim}(\omega_{\text{meas}}, \omega_x) \equiv \text{sim}(\Delta\omega) \propto \exp(\Delta\omega^2) \tag{3.32}$$

Here, observation z is represented by the actual bearing measurement ω_{meas} and ω_x is calculated geometrically from the hypothesis x given the map m.

Early integration of line percepts were crude due to the limited processing power. A line percept only contains information about a single point in robot coordinates that lies on a line; line orientation is not included. Usually, quite a few of these line points are detected in an image. The sensor model assumes these to be independent and a random selection of three such points is used for localization to keep the computational burden manageable [Röfer and Jüngel, 2004]. This is, of course, an assumption that discards much of the information inherent in perceiving a line in the image. Recent work in the GermanTeam uses the Roberts operator to determine the orientation of lines associated with a single point and aggregates these for more accurate estimation of the line's orientation [Röfer et al., 2006]. In both cases, the robot's distance to the line is used for localization. Much of the expensive geometric calculations necessary can be pre-computed and stored in look up tables.

Although generated from the same camera image, all percepts are integrated as if they were independent. This can be thought of as fusing the information of a number of sensors. The weight of a particle is thus calculated as the product of the probabilities associated with the various observations of sensors i:

$$w_t^{[n]} = \prod_i p_i(z_t^{[i]}|x_t, m) \tag{3.33}$$

Importance Weighting and Resampling

Contrary to the MCL in Fig. 3.4, the particles' weights are not reset during resampling. Instead, the new particles inherit the weight of their parents. Similarly, during sensor update, the weight is calculated using the sensor model and the current weight associated with the particle. A low pass filter is employed to make sure that occasional sensor misreadings do not have a negative effect on the particle distribution. Such a sensor error can otherwise wipe out a great number of particles and thus degrade the localization. While this low pass filter makes the filter less reactive, it also makes it more robust against false positives.

Templates

The concept of templates is similar to that of sensor resetting localization. Templates are robot poses generated directly from sensor data. Since a single landmark percept does not yield enough information, percepts are stored and propagated using the motion model to generate templates when sufficient information is available. These templates are injected into the particle distribution.

3.6.3 Practical Considerations

The two main enhancements over the text book version of MCL, namely the use of templates and the filtering of importance weight, are attributed to the small number of particles used in our implementation. The particle count does not necessarily have to be fixed. Kwok et al. show how the number of particles can be adjusted based on the approximation quality [Kwok et al., 2003]. While this helps reduce the number of particles necessary when the robot is well localized, it does not solve the problem of what needs to be done if the localization uncertainty is high. Typically, the total number of particles is limited to the average time needed per sample to integrate sensor and control data (see [Kwok et al., 2004] for an approach that aims to overcome this limitation). A small particle count has several disadvantages, which this work tries to remedy by better modeling motion and sensing. In our case, the particle count of 100 is used in a three dimensional state space. As already discussed, using few particles results in difficulties with global localization and the recovery after being kidnapped. Furthermore, the noise introduced during motion update must be limited – reducing robustness – to ensure convergence of the distribution. Lastly, although not directly related to the implementation of the MCL, but more to the process of perception as a whole, the flow of information used for the observation update depends on the task the robot is currently pursuing. If it is chasing the ball, its attention focuses on the ball, leaving little to no time for the robot to look around and detect landmarks. This results in the robot sometimes moving on the field without sensing anything relevant for localization, effectively reducing the localization to dead reckoning. Such situations are dangerous as the robot is often pushed by other robots fighting for the ball, while at the same time no percepts are available for it to correct its belief. As the ball is often kicked after the robot has reached it, localization errors can have fatal results. The above issues will be addressed in the chapters on proprioceptive motion modeling (Chapter 5) and on negative evidence modeling (Chapter 6).

3.7 Characterization of Particle Distributions

To benchmark localization approaches, the particle distribution is analyzed and quantitative measures are used to describe the quality and characteristics of the current localization. This is also important for the robot to determine if it is lost or well localized when running autonomously.

Any such benchmark is based on the development of the particle distribution during an experimental run. Since the distributions are randomly initialized and random noise is added in each time step, no two runs are identical. For large numbers of particles, however, the values of derived observables converge toward the same value.

3.7.1 Localization Error and Ground Truth

Ground truth is a term borrowed from cartography and is used to describe the true position of the robot within its environment. The true position is usually not observable by an autonomous robot and is determined by an external observer, e.g., by an external camera system [Dahm et al., 2005]. In the simulator used by the GermanTeam (SimRobotXP), ground truth is provided by the *oracle* [Laue et al., 2006]. In the comparative study of localization techniques mentioned earlier [Gutmann et al., 1998; Gutmann and Fox, 2002], the distance of localization to a set of discrete to reference points is used to benchmark the quality of localization. This measure assumes the localization module to return a single robot pose and is unable to differentiate multiple hypotheses. I chose to use the average position error of the particle distribution to measure accuracy as it also takes into account how much the distribution is spread out:

$$\Delta r = \frac{1}{N} \sum_{i=1}^{N} \left| \begin{pmatrix} x_i \\ y_i \end{pmatrix} - \begin{pmatrix} x_{\text{true}} \\ y_{\text{true}} \end{pmatrix} \right| \tag{3.34}$$

Likewise, the average orientation error is calculated. In experiments, I used points of reference on the field (measured manually) and in some cases simulation.

As it requires external sensing, the average localization error is not available to robots in-game.

3.7.2 Statistical Moments

Standard deviation σ, the second statistical moment, or variance σ^2 can be used to describe the level of uncertainty associated with a particle distribution

(see Section 3.2.1). In case of a multivariate probability distribution, the variance becomes the covariance matrix (Eqn. 3.21). Describing uncertainty in this way is meaningful only if the underlying probability distribution is Gaussian. For non-Gaussian distributions, standard deviation is still useful to estimate the uncertainty in many situations. In case of multiple well defined hypotheses, the standard deviation remains at a high value and does not acknowledge the fact that the hypotheses themselves are well defined. It only ever becomes small if the localization has converged toward a single pose or a confined area, but even then the distribution shapes typically encountered are not always well described by standard deviation (Fig. 3.7). Using higher order statistical moments may prove helpful in this context.

Standard deviation was used to some extend during games to decide if the robot was localized well enough to carry on what it was doing or if it needed to re-localize.

3.7.3 Entropy

Examining a probability distribution from an information theoretical point of view, the concept of *entropy* is quite useful. It describes the expected information that the distribution carries and is defined as follows (Eqn. 3.22):

$$H_p(x) = E(-\log_2 p(s)) \tag{3.35}$$

$$= -\sum_s p(s) \log_2 p(s) \tag{3.36}$$

As before, the sum becomes an integral over the respective range of s for continuous variables. Entropy is always a function of the system it describes and numerical values must be related to the maximum entropy of that particular system. In the case of a robot trying to localize, low entropy values denote that the robot is relatively certain about its whereabouts, whereas high levels mean that it does not know where it is. *Information gain* brings about a decrease in entropy.

In a particle filter, the probability density $p(s)$ is represented by the particle density. In order to calculate $p(s)$ and thus the sum, the state space $s = (x, y, \theta)$ is discretized into equal sized cells. A probability histogram over the grid cells is calculated by counting the number of particles per cell n_i and normalizing using the total number of particles N. The probability of cell i is thus given by $p(s_i) = n_i/N$ and Eqn. 3.36 becomes a sum over all cells I:

$$H_p(s) = -\sum_i^I p(s_i) \log_2 p(s_i) \tag{3.37}$$

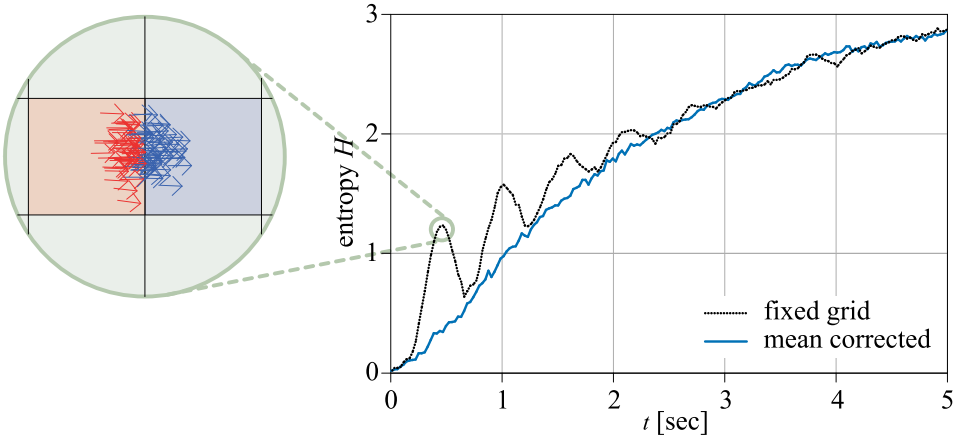


Figure 3.6: Calculating the entropy of a particle distribution. Entropy calculated using a fixed grid and a grid centered around the mean of the particle distribution. The particle distribution represents a robot moving forward (see Section 5.5 for details). *Highlighted:* Cluster of particles crossing the boundary between two cells. Half the particles are in the left, the other half is in the right cell. The computed entropy rises even so the uncertainty remains at a comparable level. Centering the grid around the distribution mean eliminates these artifacts.

One important result of this is that the numerical value of the entropy of the PDF is also a function to the size of the grid cells. The maximum value of entropy occurs when the particles are equally distributed over the entire state space: $H_{\max}(s) \approx -\log_2 I^{-1}$ with $p(s_i) = N/(I \cdot N)$. The solution is only approximate as the probabilities can only take on integer multiple values of the inverse of the particle count $1/N$. I chose the number of cells to be smaller than the number of particles.

The minimum entropy value of zero is reached when all particles are confined in a single cell; within this cell, the particles may all have the same values (x, y, θ), but may also be equally distributed.

If a static grid is used, artifacts as shown in Fig. 3.6 can appear: if the particle distribution's standard deviation is of the order of the size of the grid cells and a static grid is used in the computation, the entropy rises as the particles cross a cell border, although the uncertainty can be regarded as being constant. This is addressed by centering the grid around the mean of the particle distribution. For above mentioned distribution representing a moving robot, most particles thus remain within the center cell and only move into adjacent cells if the uncertainty has truly increased. For spread out distributions, the artifact becomes negligible as can be seen on the right of Fig. 3.6.

In the experiments, grid cell dimensions of (50 cm, 50 cm, 10°) were used. In some experiments, smaller grid dimensions were chosen if the state space was known to be limited to a part of the field. Furthermore, separately examining the position and orientation uncertainty of the robot sometimes provides further insights. It can help differentiate particle configurations that have some sort of order, which may otherwise be missed.

Entropy is used to describe uncertainty in all of our experiments.

3.7.4 Uncertainty with Respect to a Task

In Fig. 3.7, particle distributions are shown that are not well described by the characterizations covered so far.

Let us consider the task of localization using triangulation on a RoboCup field. It requires the robot to subsequently look at different landmarks. If the distance to landmarks is used, three landmarks are sufficient. (In the GermanTeam MCL implementation, only bearing is used, and the robot therefore needs to detect four individual landmarks to be able to localize.) An Aibo on a RoboCup field can *pan* its head and thus scan approximately 230° of its surroundings, which yields sufficient information for triangulation in most cases. In more complex environments, where landmarks are not all close to the horizon, where there are walls and obstacles, and where landmarks may be occluded, actively gazing at landmarks is more efficient. If localization is reasonably well, actively gazing at landmarks can be used to efficiently and continuously confirm the current localization. In the case of completely uniform PDF, active vision is not possible and the robot has to resort to hard wired search patterns. If localization is poor but not completely uncertain, it is desirable to select a gaze direction that is expected to maximize information gain [Cassandra et al., 1996].

Determining the expected information gain of a sensing action is not a trivial task and requires estimating robot action and the predicting the effect of expected future sensor readings. I suggest a new approach to describing the uncertainty by estimating the uncertainty with respect to a given point in the robot's environment. As the bearing to landmarks is used for localization (Fig. 6.1), the angle of particles to landmarks can be used as an indicator for localization with respect to a particular landmark. More specifically, the standard deviation of the bearing of all particles to landmarks can be calculated. The landmark with the highest bearing variance is the landmark that yields the greatest information gain if the robot were to look at it, i.e., the actual and the expected angle differ the most for this landmark.

In the triangulation task, such an approach can be used to determine which landmark to next look at. Take the distribution in Fig. 3.7 b), which can be

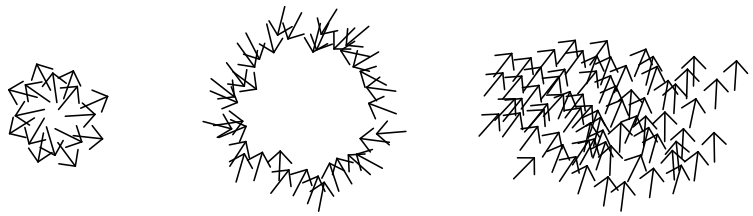


Figure 3.7: Particle distributions that are not easily identified using common characterization techniques. a) The spacial localization is good but the orientation of the robot is completely uncertain. b) Circular configuration of particles as it occurs when distance information to a landmark is used. c) Particles aligned toward a landmark when bearing is used for localization.

understood as an intermediate step in the process of triangulation. Such a circular distribution would result from the robot having perceived a landmark in the center of the circle. Calculating the variance with respect to available landmarks, the landmark in the center of the circle would be the one yielding the least information gain (as the robot has just looked at it) and a landmark outside the circle would be chosen.

This approach has interesting implications on the process of perception and active vision. Should the probability distribution diverge in the process of triangulation (e.g., by noise introduced by the robot moving), the landmark yielding the greatest information gain may change over time. This causes the robot only to look at a landmark until it has seen it and until it has drawn the maximum information from it. This is particularly interesting as the rate at which particle weights are adjusted is limited and thus requires the robot to look at a landmark for some time for the weights to change. The camera control system can thus use feedback from the localization module (the decrease in uncertainty) to determine where and for how long to next direct the robot's gaze and with little information about the actual filter implementation.

The evaluation of this view on uncertainty is ongoing work and not covered in this dissertation.

Performance in Dynamic Environments

As the experimental setup becomes less restricted and the environment is allowed to be dynamic, objective benchmarking becomes increasingly difficult. In our case, the active vision based head control alone makes it hard to compare two experimental runs as the localization will be based on slightly different sensor data, even if the setup is (almost) identical.

Ultimately, the overall performance of the system is determined in a soccer game. In such a complex game, it is difficult to identify the effect of individual changes. We therefore focus on keeping the environment as controlled as possible to emphasize the effects of the proposed improvements.

3.8 Summary

Many approaches to mobile robot control and to mobile robot localization have been proposed. The limitations of model-based control lead to sensor-centric, behavior-based control [Brooks, 1986]. While successful in simple tasks, achieving more complex tasks requires some higher order planning and control found in hybrid architectures [Arkin, 1998]. In more recent years, probabilistic approaches have been met with increasing interest and of these, Monte Carlo localization has been the most successful in many localization tasks [Thrun et al., 2005]. In probabilistic modeling, the robot pose is modeled not as a single, certain set of values, but by a probability density function. This PDF accounts for the fact that localization is often uncertain and ambiguities occur frequently. MCL models the distribution by a set of particles, each representing a hypothetical robot pose. The density of particles is proportional to the probability of the robot pose. The GermanTeam uses Monte Carlo localization with a comparatively small particle count due to the limited computational resources available on the Sony Aibo. While a large number of particle makes MCL robust against localization failure and unforeseen, un-modeled events, being restricted to few particles can result in localization errors and reduced robustness.

This thesis focuses on enhancing the motion model and sensor model, thus making the probability distribution more responsive and improving MCL's robustness. This is preceded by a behavior based approach to collision avoidance, which later serves as the basis to better model robot sensing.

Chapter 4

Obstacle Modeling and Collision Avoidance

4.1 Introduction

In this chapter, I will present a complete system for collision avoidance for a mobile robot. This system serves two purposes: the obvious one is to allow the robot to safely navigate in the presence of obstacles. The other purpose is to provide the means to detect occlusions of parts of the environment. This will later be used to improve sensor modeling, while this chapter focuses on obstacle modeling and avoidance.

While collision avoidance is important for most mobile robot applications, little work has been done on it in the Sony League of RoboCup. This is partially due to the lack of static obstacles, the only obstacles being other players. Obstacle avoidance within a team is often achieved by communicating the current position and intention. The impact of collisions is severe as the robot is greatly slowed down and actions are not executed as desired, resulting in unpredictable outcomes. Furthermore, collisions often leave the robot badly localized, leading to undesirable actions such as kicking the ball in the wrong direction.

We therefore built a system for goal-directed obstacle avoidance consisting of the following modules:

- vision system (yields: obstacles percept)
- modeling layer (yields: obstacles model)
- robot behavior module (yields: motion commands)

The system uses monocular vision data with a limited field of view. Obstacles are detected on a level surface of known color(s). This offers an efficient

method to detect obstacles and to estimate the robot's distance to obstacles.

A two-layer visual sonar approach is used to model free space in the vicinity of the robot. A map of close obstacles is stored in a radial proximity map, actual proximity measurements are stored in cartesian coordinates and are coupled to the radial map. This helps keep the number of stored past measurements limited and allows efficient updating of the model. Past measurements are used to update the model when no new sensor readings are available. The model integrates both current and recent vision information.

The two-layer model is used to achieve accurate and fast obstacle avoidance in a dynamic environment. The obstacle model is accessed by means of analysis functions that calculate properties relevant to specific tasks. The overall system was used with much success in the RoboCup 2003 obstacle avoidance challenge in the Sony Four Legged League. Our robot was able to finish the course in the shortest time, being almost twice as fast as the runner up.

The system in its entirety enables the robot to detect unknown obstacles and reliably avoid them while advancing toward a target.

Outline In the related work section, I will give a brief survey of sensors used in the context of robot navigation and also describe existing approaches to obstacle avoidance. The obstacle avoidance system is then described, covering the areas perception, modeling, and control. The chapter is completed by giving experimental results.

4.1.1 Sensors

Obstacle avoidance is often achieved by reactive control based on direct sensing of the environment. I will therefore first give a brief overview of the sensors typically used in robot navigation before I cover work describing robot control based on these sensors. A sensor is a device that responds to a physical stimulus (heat, light, sound, pressure, motion, etc.), and produces a corresponding electrical signal. The sensing range describes the physical volume, area, or distance in which the stimulus must occur for the sensor to detect it.

The following sensors directly measure the distance from the sensor to the next object. They are *active* because they emit sound or electro-magnetic waves in order to detect reflections caused by nearby objects.

Sonar

Measuring distances using sonar is based on the speed of sound v_m in a given medium (most commonly air or water). Active sonar creates a pulse of sound, often called a *ping*, and then waits for the echo reflected off an object. The time traveled t of the signal (round trip) is measured and used to calculate the

distance to the object $x = \frac{1}{2}(v_m \cdot t)$. The bearing to objects can be calculated using two or more microphones and measuring the differences in run time of the return signal.

Since sound travels as waves, the volume covered by the emitted sound is of roughly conical shape; it is thus termed *sensing cone*. Its true shape is highly complex. The energy of the sound wave decreases with distance to the sound source. This limits the range of the sonar, as the sound has to travel to the obstacle and back while the return signal has to be strong enough for the microphone to detect.

The sensing range of sonar used in robots lies between centimeters and several meters. In naval applications and underwater robots, range can go up to kilometers. Typically, an array of sensors is used, each measuring the range in a sector around the robot. Such an array may be mounted at the front of the robot with the sensors measuring the free space in adjacent regions. A cylindrical layout is called a sonar ring. When using an array, cross-talk between sensors may occur leading to erroneous distance measurements (one sensor detecting the ping of a neighboring sensor). Another source of systematic error is caused by sound hitting a surface under a shallow angle, resulting in the sound bouncing off away from the robot yielding no echo. Such specular reflections cause the sensor to miss an object that is in sensing range.

Laser Range Finder

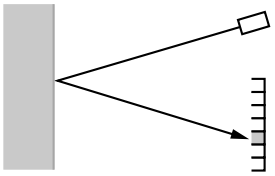
Laser range finders use a laser beam in order to determine the distance to an opaque object. This coherent, highly focused pulse of light is emitted and the time it takes for the pulse to travel to the object and back is measured. Although the laser beam itself is one dimensional, the sensor can be modified relatively easy to provide serial 2D and even 3D scans at relatively high update rates. This can be done because (a) the beam can be re-directed easily using a mirror and (b) the time of flight is extremely short and the speed of light is many orders of magnitude greater than mechanical movements of the robot.

Similar to sonar, the laser beam may be reflected away from the sensor if it hits a reflective object at a shallow angle. Furthermore, certain materials such as glass are translucent to laser light and therefore invisible to the range finder. Lastly, the laser beam is very narrow and it spreads out much less than sonar. This means that there is essentially no sensing cone but only the laser beam. Typically in a 2D laser scan, beams are in a plane parallel to the ground; obstacles below or above that plane remain undetected. The range of laser range finders used in indoor robotics is of the order of meters with an error of the order of centimeters.

Laser range finders are related to radar, however the wavelength of the light used in radar is especially sensitive to metal objects.

Infrared Distance Sensor

Distance is measured by sending out a continuous infrared (IR) beam by the emitter onto a surface where it is reflected. The reflection is detected on a linear array of photo-detectors sensitive to the particular wave-length.



The output of the sensor is non-linear. The distance to the object is calculated using trigonometry. Below a certain distance, the measurements tend to rise again, so caution is advised when dealing with IR distance measurements.

The Sony Aibo ERS-7 has three IR sensors: short-range head sensor (sensing range: 50−500 mm), long-range head sensor (sensing range: 200−1500 mm), and chest sensor (sensing range: 100 − 900 mm). While the IR beam of the chest sensor points forward when the robot is standing up using the Sony walking engine, it is directed at a point on the ground about 20 cm away from the robot when the GermanTeam walking engine is used. If the ball is placed right in front of the robot almost touching it, this sensor produces a very large distance reading (upper limit of its sensing range), as the sensor is unable to detect the reflection. This sensor response can be used to determine if the ball is below the robot's head. In this case, the robot is unable to see the ball and this additional information can be used to trigger a ball grabbing motion.

The head IR sensors are only used in preliminary experiments as vision data provides richer data.

Touch Sensitive Bumpers

Touch sensitive bumpers are often used to detect contact or collisions of the robot with its environment. Bumpers are simple and inexpensive, button-like sensors that can detect if pressure is exerted on the sensor. The Aibo has touch sensitive, spherical bumpers in its feet to detect if the feet are touching the ground. Output of such sensors can be on/off or an analog signal if sensor is force sensitive. Touch sensors can be combined in a circular configuration to determine the direction of the exerted pressure.

Monoscopic Vision

Digital cameras are passive sensors and rely on sufficient external lighting. In contrast to the above sensors, obstacles cannot be detected directly by the

sensor itself. The camera image has to be analyzed and obstacles need to be detected in it. In general, this is a difficult problem that requires knowledge about the environment. The *figure/ground problem* needs to be solved, i.e., separating objects in the foreground from the background (a figure in this context is a familiar configuration). In the robot "Polly," the *background texture constraint* is used to distinguish objects from the ground in the case of uniform ground texture and/or color [Horswill, 1993, 1994]. Having separated objects from the image background, the distance to the object needs to be calculated. Under the *ground plane constraint*, i.e., robot and obstacles reside on a plane, the height in the camera image correlates to a distance to the robot. Distances can be stored in a radial depth map similar to the output of a sonar ring. Because of this analogy, such a map has become known as *visual sonar.* In the case of the RoboCup environment, free space around the robot is associated with green, the color of the ground, whereas non-green colored pixels are associated with obstacles (field lines need special treatment). While this is often done using a color segmentation, Lorigo et al. [1997] use the bottom of the image to continuously determine what the ground looks like.

The sensing range of a camera is called viewing frustum and is determined by its opening angle. Cameras vary in resolution, color resolution (bits/channel), frame rate, opening angle, quality of the optical elements used, etc. The Sony Aibo ERS-7 uses a fixed-focus camera of 55° opening angle (horizontal) and a resolution of 208×160 YUV with 8 bits/channel at 33 Hz.

Omnidirectional Vision

Omnidirectional vision is achieved by a conical mirror [Benosman and Kang, 2001] (Fig. 4.1). The reflections in such a mirror provide a 360° view around the symmetry axis of the mirror. The mirror may be curved resulting in different projection properties, e.g., relative spacial resolution and maximum sensing range. Due to the radial symmetry of the mirror, areas that are projected onto the center of the camera image (i.e., reflected near the tip of the mirror) have a higher resolution than those near the image borders. Using omni-vision, a visual sonar can easily be constructed from every single camera image without the need of remembering previous sensor readings [Weigel et al., 2002; von Hundelshausen et al., 2005].

The mirror can be set up to create a panorama of the surroundings. This is used for video conferencing where the camera is placed on a table top and is able to capture all participants. For mobile robot application, the curvature is designed for the camera image to cover the area directly surrounding the robot, extending toward the horizon and slightly above it. One drawback of this projection is that much of the camera's resolution is used up for the areas directly surrounding the robot, which limits the resolution of further away

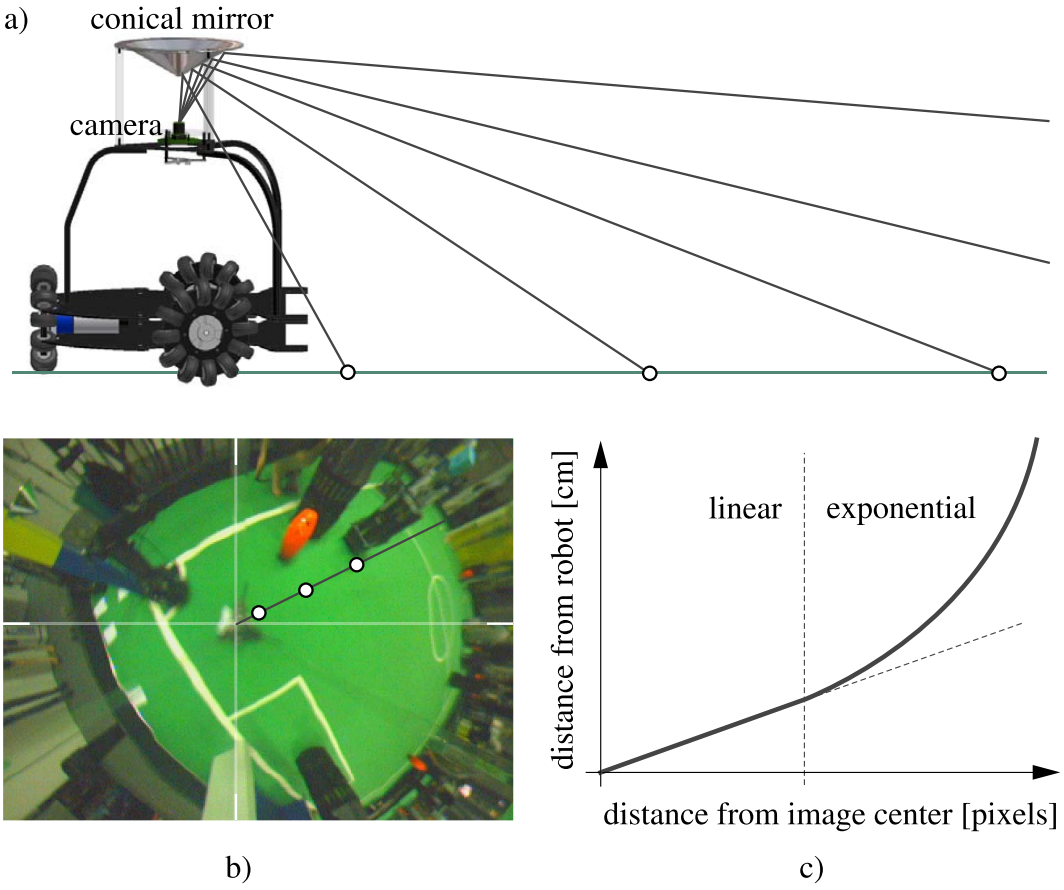


Figure 4.1: Middle-Size Robot of the FU-Fighters using omni-vision. a) The camera is pointing upwards toward the mirror, rays of light are shown. b) Actual camera image. c) Transformation necessary to convert the distance from the image center into distance from robot center. (Diagram and photo courtesy of Felix von Hundelshausen, Freie Universität Berlin.)

areas. The setup as a whole needs to be well calibrated as small imperfections and misalignments lead to images being distorted in unpredictable ways.

The inverse of the projection is calculated to determine coordinates of objects in cartesian space relative to the robot. A further assumption is that all objects lie flat on the ground within a circle around the robot. This results in distortions for objects which are raised (three-dimensional), e.g., the ball becomes an ellipse (see Fig. 4.1).

Stereoscopic Vision

Stereo vision uses the disparity between two images to estimate the distance to an object [Molton and Brady, 2000; Davison and Murray, 1998], creating a depth map. Distance estimation works well for small distances; for large distances, the disparity is small, which leads to an unfavorable signal to noise ratio. Furthermore, objects of high contrast and rich texture are necessary to find corresponding pixels (*correspondence problem*); in images where this is not the case (e.g., when facing an uniformly illuminated white wall), little or no depth information can be extracted. Depth data is not per se continuous over the field of view, as only the distance to features can be calculated. Interpolation may be used to fill in the areas between features.

4.1.2 Modeling and Control

This section gives examples of how sensor data can be used to model the proximity of the robot and to achieve robot control.

Potential Field

In analogy to physics, reactive collision avoidance is achieved by following the resulting force vector $\vec{F}_{\text{res}}(\vec{r}) = -\nabla\varphi(\vec{r})$ generated by a potential field $\varphi(\vec{r})$ at a position $\vec{r}$. The agent follows the force vector determined by the potential field, which thus determines the path taken. The target location of the navigation task is modeled as a minimum in the potential field, whereas obstacles are peaks in the potential field, resulting in a repellent force. A potential field can be thought of as a surface and the agent being a ball rolling down towards the field's minimum. Potential fields are generally composed of a number of elementary potential fields by superposition.

Potential fields for robot control were first proposed by [Khatib, 1986]; they work well for small numbers of obstacles near the robot and produce smooth robot trajectories. In more complex scenarios, global navigation is difficult to achieve using potential field as the robot tends to get stuck in local minimum traps [Arkin, 1998; Latombe, 1991]. Ideally, if robot inertia is not considered, the robot speed becomes zero as the robot converges to equilibrium position with $|\vec{F}_{\text{res}}| = 0$. In a dynamic system, however, the robot overshoots the minimum and starts oscillating around it, which can make it hard to detect that the robot is actually trapped.

Virtual Force Field

The Virtual Force Field (VFF) approach combines a certainty grid for obstacle representation with potential fields for navigation [Borenstein and Koren,

1989]. Every time an obstacle is detected by the sensor, the value corresponding cell in the two-dimensional certainty grid is incremented. This can also be thought of as a 2D histogram of the measurements. Since this map stores information about whether or not a cell is occupied, the term *occupancy grid* is also used. Using the grid, a force vector $\vec{F}_{\text{res}}$ is generated by superposition of forces $\vec{F}_{ij}$ of all cells ij, $\vec{F}_{\text{res}} = \sum_{\text{cells}} \vec{F}_{ij}$. The force decreases with distance from the obstacle to the robot with d^{-2}, i.e., close obstacles exert a larger force than far away ones. Also, the force is proportional to the value stored in c_{ij}, therefore obstacles that the robot is certain about produce a greater force. Since misreadings are common but randomly distributed, clusters of filled cells representing actual obstacles develop over time, resulting in a strong repellent force. Directing the agent towards a goal is achieved by adding a force $\vec{F}_{\text{target}}$ of constant length pointing towards the goal.

Heuristics can be introduced to determine if the robot has run into a local minimum trap (other more complex obstacle configurations can also be identified). Once a situation is detected, the robot switches to a recovery mode. In this mode, it follows the closest wall to circumvent the obstacle and reach a position from which it can resume VFF navigation. Wall following is easily implemented as a sensor-based behavior if an appropriate sensor can continuously measure the distance to a wall when the robot is moving parallel to it (sonar, laser range finder, omni camera, etc.). In the special case of a robot in a structured indoor environment, the vanishing point of detected lines in a camera image can be used to estimate the distance to walls and steer the robot [Schuster et al., 1993]. In a labyrinth-like setting, the robot chooses whether it will follow the wall to its left or to its right in order to reach the goal. The robot then sticks to a choice when it later encounters walls.

The VFF approach generally works well, but the robot was observed to have trouble passing through doors. When trying to navigate through a door, both corners push the robot away and control is prone to oscillations. The *Vector Field Histogram* extends the VFF approach by generating a circular histogram of the free space around the robot [Borenstein and Koren, 1991]. The occupancy grid is still maintained, but an intermediate data reduction step is introduced, creating a polar histogram of the occupancy grid. Obstacles in the immediate vicinity of the robot produce larger magnitudes and the weight of each cell is proportional to the certainty associated to it. Robot control seeks valleys in the radial map to steer the robot towards the goal while avoiding areas of close and certain obstacles.

The two methods mix uncertainty and actual spacial information when modeling the environment. Obstacles which have been detected many times have a larger effect than obstacles that have appeared only recently. This works reasonably well in static environments and with panoramic sensors, where

obstacles appear at the far end of the sensing range and then move towards the robot, building up confidence in the obstacle as it approaches and thus building up the repellent force. For dynamic environments and when using a limited field of view camera, this underlying assumption is no longer true and the approach is not feasible as obstacles may "pop up" into view and immediate action is called for.

Dynamic Window Approach

In the dynamic window approach to obstacle avoidance, control action is chosen in velocity space. For the synchro drive robot used in the experiments, translational speed v and rotational speed ω can be used to describe robot motion. These two control parameters result in unique circular robot trajectories. Constraints and preferences are imposed on the possible parameters.

The maximum change of velocities is constrained by the torque limits of the robot's motors. Furthermore, using a model of obstacles around the robot and assuming a maximum deceleration, combinations of v, ω that would result in future collisions with obstacles are identified and discarded. Taking into account the robot's current v_t, ω_t further limits the range of new control commands v_{t+1}, ω_{t+1}. This area around the current speeds in the space of velocities is called *dynamic window*, where *dynamic* refers to the dynamics of the robot and not to the size and position of the window.

Now that the possible range of control commands has been reduced, *preferences* are used to determine a control action. Preferences are expressed by means of utility functions; these are:

target heading angle to target: heading(v, ω)

clearance expresses a preference towards paths that will not intersect obstacles at all, regardless of robot speed: dist(v, ω)

velocity higher translational velocities are preferred over lower: velocity(v, ω)

The overall preference is given by the weighted sum of these three functions. Heading has by far the greatest weight, whereas the clearance and velocity term serve to make control more robust and help the robot circumnavigate obstacles.

Given the hard constraints that forbid certain v, ω, the remaining combinations in the dynamic window are evaluated and the control action with the highest utility is chosen. The approach was successfully tested and used in the robot "Rhino" [Buhmann et al., 1995; Fox et al., 1997].

Model Based Dynamic Window The model based dynamic window approach, μDWA, extends the dynamic window approach by generating virtual sensor readings using a simulation of a proximity sensor that is able to detect all static obstacles given the current world model and a map [Fox et al., 1998b].

For the museum tour guide robot "Minerva" to safely operate in the "Deutsches Museum Bonn," navigation needed to reliably avoid running into visitors and into exhibits. Minerva was equipped with the following sensors: stereo camera, sonar, tactile sensors (bumpers), laser range finders, and infrared sensors. Although the robot was equipped with this abundance of sensors, some obstacles were still difficult to detect (see above section on sensors). To be able to deal with such obstacles, a hybrid-approach that integrates sensor readings and model-based predictions was devised. It extends the sensor-based dynamic window approach by integrating world model-based information about the obstacles into robot control. A virtual sensor is used, which generates proximity sensor readings based on the current, potentially uncertain, localization. Taking this uncertainty into account, the virtual distance measurement is probabilistically distributed. From this distribution, which varies with localization uncertainty, a distance is selected that has a high probability of being at the lower end of the distance spectrum, yielding a conservative distance estimate and allowing safe locomotion. This method is also known by the name of curvature velocity method [Simmons, 1996].

4.1.3 Related Work

Let us take a brief look at what can kind of robot navigation can be achieved if processing power is a non-issue and multiple sensors can be used to offer the best possible sensory input:

Stanley

Stanley, a stock, Diesel-powered Volkswagen Tuareg R5 equipped with various sensors and computers, developed by Thrun et al. at Stanford University, recently won the 2005 DARPA[1] Grand Challenge. Stanley was the first of five robots to finish the 212 km course through the Mojave Desert in 6h53m. It was equipped with the following sensors: 5 laser range finders (short range), radar (long range), monocular video camera, GPS sensor with 20 cm resolution for pose estimation, measurements of wheel speed for pose estimation (odometer), 6DOF inertial measurement unit, and a GPS compass that generates 2DOF balance information from two separate GPS antennas.

[1] Defence Advanced Research Projects Agency

Figure 4.2: Stanley, the winning robot of the 2005 DARPA Grand Challenge and one of five robots to finish the 212 km course through the Mojave Desert. Note the five laser range finders mounted on the roof of the car, each tilted at a different angle. (Photo courtesy of Stanford Racing.)

The sensor data is processed by 6 Pentium M motherboards in a rugged rack mount unit. Data is sampled from instruments at rates varying from 10 Hz to 100 Hz. A battery-backed, electronically-controlled power system is used.

As GPS and, for that matter, any single sensor is not accurate enough to steer the car across the course, the sensor data of all sensors is combined. This process is called *sensor fusion*. The 2D laser range finders are tilted at different angles, pointing at the ground at various distances from the robot. As the car moves on, the 3D scan of the ground is completed over time, incorporating data from the scanners and the radar and also using the inertial measurement unit to compensate for movement of the car itself. As this only covers the vicinity of the robot, additional long range information is extracted from the camera images.

The details of control are too complex to cover in detail and have not been published to date. In brief, the robot learns what drivable surfaces look like and searches the area ahead of it to find the road.

RoboCup

Robots used in RoboCup have much more limited sensing and processing capabilities. To somewhat compensate this, the environment is kept relatively stable and well defined. Color coding and localization beacons further help the robot. However, navigation represents only one aspect in this competitive multi-robot setting.

In all leagues, collisions are not only disadvantageous, they are also penalized. But as collisions are hard to detect for robots, a collision is defined as continuously touching or pushing another robot for some time (two seconds in the case of the Sony Four-Legged League).

Compared to the Sony Four-Legged League, the sensor data available to the robot in Small-Size and Middle-Size League is much richer and allows more elaborate, global world modeling and path planning techniques. One or more cameras mounted above the field give the robots in the Small-Sized League a complete view of their environment.

The OMNI RoboCup small-size team segments an image based on background color to find free space around the robot using an omni-directional camera [Sekimori et al., 2002], representing the vicinity of the robot in a polar diagram. However, no memory of past measurements is necessary as the environment can be monitored continuously. Obstacle avoidance is achieved by trying to steer the robot in the target direction. If there are obstacles in that direction, the robot slows down and turns toward where there is free space for it to move on. This is done taking into account the current robot velocity. Using rapidly exploring randomized trees, the vicinity of the robot can be explored and, as the tree grows, a path towards a goal can be found [Bruce and Veloso, 2002]. In 2001, the FU-Fighters used dynamic programming on a grid representing obstacles and free space on the field to find an optimal path [Rojas et al., 2002]. This is embedded in a hierarchical behavior architecture, where different layers work at different time scales [Behnke and Rojas, 2000]. Much of this was also used in the Middle Size team of the FU. The path planning approach was later refined to accommodate for the dynamics of the environment and the fact that the area around the robot is both more accessible to the robot and also more important than far away regions. The environment is thus represented by a multi resolution grid, using fine grid cells in the direct vicinity of the robot and a coarse grid farther away [Behnke, 2004].

In early days of the Middle-Size League, the field was surrounded by a border and several successful teams like CS Freiburg [Weigel et al., 2002] used laser range finders. Detecting the borders allowed efficient localization. A potential field approach was used for navigation and obstacle avoidance as the world model available to the robot is highly accurate. Potential fields are also used to “dribble” the ball and at the same time avoid obstacles on its way

towards the goal [Damas et al., 2003]. Recent work within the GermanTeam aims at using potential fields to create complex robot behavior [Laue and Röfer, 2005], including obstacle avoidance, strategic positioning, and path planning. While the results are promising, the problem of having to maintain a global world model is not solved, making the approach susceptible to errors if the model is imperfect.

With the omission of the field borders in the Middle-Size League, omni-vision cameras became increasingly popular. The omni-vision camera can be used to produce output similar to that of a 2D sonar or laser range finder [Menegatti et al., 2005]: instead of detecting reflective surfaces, chromatic transitions in the image are extracted and used for world modeling.

Collisions within a team can be avoided by role assignment. Roles are either static or may change with respect to the current game situation. A global reference frame is helpful to determine actions within the team [Behnke and Rojas, 2000]. Inter-robot communication can help to disambiguate insufficient information about the robot's environment [Röfer et al., 2005]. The robot closest to the ball becomes the striker and "tells" the other robots to back off, avoiding robot clutter and allowing strategic positioning. In 2001, the robots of the UNSW team used a backing off behavior when visually detecting another team mate; the robot emits a high pitched sound to tell the other robot what it is doing and where it is [Chen et al., 2001].

Using the vision data without building a model, only very basic obstacle avoidance is achievable since the opening angle of the camera is small, making it difficult for the robot to see enough of its environment to allow for safe navigation (see next section). To also allow the robot to localize and pursue a goal, a model of the environment is necessary. *Visual sonar* builds up a radial model of obstacles close to the robot [Lenser and Veloso, 2003]. Free space around the robot is detected and this information is integrated into the model, which serves as a memory when the robot is directing its gaze away from regions relevant for collision avoidance. The approach presented here extends this concept by adding a second model layer to allow more efficient robot control.

4.1.4 Preliminary Experiments

Two preliminary experiments are described here to outline what can be achieved using even simple, sensor-based approaches, and to motivate the use of the later presented system for collision avoidance.

Modeling the robot's surrounding is particularly important in real world robotics and especially in cases where the robot has no sensors to directly detect obstacles.

Simple Obstacle Avoidance Using Minimal Sensor Data

For initial testing and benchmarking, a behavior similar to that of a 2nd order Braitenberg vehicle was implemented (light seeker/avoider, [Braitenberg, 1984]). It makes use of the color coding of the robot's environment: unoccupied floor (i.e., free space) on a soccer field is green while obstacles are colors other than green. In analogy to the light sensors of the Braitenberg vehicle, the camera image is divided in two halves. For each half, the number of green pixels is calculated and normalized to the total number of green pixels in the camera image. The camera itself points slightly below the horizon to make sure that the camera is looking at the floor. The robot moves forward at a constant speed. Depending on the amount of floor colored pixels encountered in the left or right half, the robot turns towards the direction where there is more free space while still moving forward. Such a behavior lets the robot wander about the field aimlessly while avoiding obstacles (such as field borders, other robots, peoples' feet). At times, however, it cannot turn quickly enough to avoid running into an obstacle.

Using the Infrared Proximity Sensor

The simple obstacle avoidance approach can be extended by using the one-dimensional infrared proximity sensor in the robot's head to control the speed of the robot. The head is tilted towards the ground such that the IR beam hits the ground at a distance of 50 cm away from the robot. Pointing the infrared sensor to the ground rather than parallel to it may at first not sound plausible, because the sensing range is not used to the fullest. It does, however, have several advantages: when pointed parallel to the ground, only obstacles that are at least as high as the vertical offset of the IR beam are detectable, whereas a beam pointed at an angle can detect even small obstacles. Furthermore, such a configuration also gives the robot the ability to detect downward steps and edges ("cliffs," e.g., the edge of a table top or the beginning of a stair case), as this will result in a larger distance reading compared to normal flat surfaces.

The robot moves forward at full speed if the distance measured equals the distance to the ground (keep in mind that the robot's head is pointed at the ground). For distances smaller or greater the robot slows down and even backs off. The robot is thus able to wander about on a table top, avoid obstacles on the table and to stay clear of the table's edge. Adding this control enables the robot to stop in its track when it happens to come too close to an object. This often happens when it is turning or when there are dynamic objects in its environment. It did, however, have problems with objects of small width because the 1D distance sensor easily misses them (e.g., the leg of a chair).

4.2 Obstacle Model

The previous example shows the limit of what can be achieved by simple sensor data based, reactive behaviors. Creating a model of the robot's surroundings is useful as it gives the robot information of what it has recently seen and it can help to avoid small obstacles that the robot's distance sensor is currently missing, but has detected recently.

The following paragraphs describe the obstacle avoidance system used by the GermanTeam in the RoboCup obstacle avoidance challenge and actual games. The goal was to find an obstacle avoidance solution best suited for the demands of the dynamic RoboCup domain. This implies that modeling obstacles is only one of the perception task the robot is required to fulfill. The robot's camera is also used to look at landmarks for localization and to track the ball. This further stresses the need of a model that serves as a short term memory as sensor data may not always be available.

The experiments were performed using a Sony Aibo ERS-210(A) robot. The robot has a 400MHz MIPS CPU and a camera delivering 8bit/channel YUV images with a resolution of 172x144. The Monte Carlo localization described in 3.6 was used; other modules not covered here such as walking engine, etc. are described in more detail in the GermanTeam 2003 team description and team report [Röfer et al., 2005].

4.2.1 Obstacle Detection

Image processing extracts features from the image called *percepts*. A percept contains information retrieved from the camera image about detected objects or features later used in the modeling modules. It only represents the information that was extracted from the current image. No long-term knowledge is stored in a percept.

The *obstacle percepts* can be seen as a set of lines on the ground representing the free space in front of the robot in the direction the robot is currently pointing its camera. Each line is described by a *near point* (white circle, green outline in Fig. 4.3) and a *far point* (red circle, white outline) on the ground relative to the robot (the grey rectangle indicates an obstacle). The lines in the percept describe segments of ground-colored lines in the image projected onto the ground. For each point, information about whether or not the point was on the image border is also stored.

To generate this percept, the image is being scanned along a grid of lines arranged perpendicular to the horizon. The grid lines have a spacing of 4°. They are subdivided into segments using a threshold edge detection algorithm. The average color of each segment is assigned to a color class based on a color look-up table. This color table is calibrated manually prior to the run. For

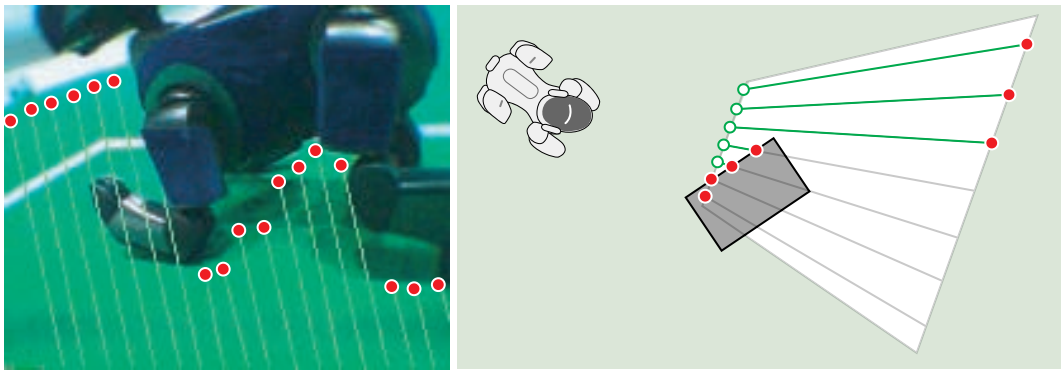


Figure 4.3: *Left:* Actual obstacle percept in an image acquired with the Aibo's camera. Lines start at the bottom of the image, extend towards the horizon and end where an obstacle is detected (note that the other robot's legs are falsely not detected as obstacle). *Right:* Schematic of the obstacle percept in a top down view.

each scan line, the bottom-most ground-colored segment is determined. If this ground-colored segment meets the bottom of the image, the starting point and the end point of the segment are transformed from the image coordinate system into the robot coordinate system and stored in the obstacles percept; if no pixel of the ground color was detected in a scan line, the point at the bottom of the line is transformed and the near point and the far point of the percept become identical.

Small gaps between two ground-colored segments of a scan line are ignored to assure robustness against sensor noise and to also assure that field lines are not interpreted as obstacles. In such a case, two neighboring segments are concatenated. The size limit for such gaps is 4 times the width of a field line in the image. This width is a function of the position of the field line in the camera image and the current gaze direction of the camera. Fig. 4.4 shows how different parts of scan lines are used to generate the obstacle percept.

The percept can be thought of as a visual sonar reading. Note, however, that the sensing range is limited in distance (both near and far) depending on the tilt of the robot's head. Similarly, only a small horizontal window of the world is accessible to the sensor. Unlike actual sonar, where the sensor model is determined by the physical measurement process, we are able to tune the vision system such that it generates very few to none false positives. In other words, the vision system can discard ambiguous readings and only produce percepts when sensor data is of sufficiently good quality. This comes at the price of an overall reduced detection rate and requires the model to be able to integrate information over time.

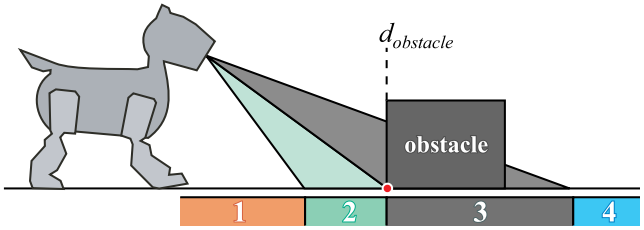


Figure 4.4: Diagram to illustrate what can be deduced from an obstacle detection. The robot detects some free space in front of it (2) and some space that is obscured by the obstacle (3). Nothing can be said about the areas outside the field of view, (1) and (4).

4.2.2 Obstacle Model

The obstacle model described here is tailored to the task of local obstacle avoidance in a dynamic environment. Local obstacle avoidance is achieved using the obstacle model's analysis functions described below. The assumption was made that some high level controller performs path planning to guide the robot globally. Certain static global setups will cause the described algorithm to fail, but such setups hardly occur in the RoboCup domain and they are a different type of problem that needs to be dealt with by higher levels of path planning. I therefore concentrate on a method to reliably steer the robot clear of obstacles while changing its course as little as possible.

A two layered model is used, consisting of two coupled maps (Fig. 4.5). In one layer, a radial representation of the free space surrounding the robot is stored. This *proximity map* stores information about free space in direction θ. It is divided into n discrete sectors ("micro sectors"). In addition to the free space per sector, the last obstacle measurement is stored as a vector in the *measurement map* of the model. This measurement is called a *representative* for that sector; there is only one representative stored per sector. Having these two closely coupled models allows for efficient updating of the model and is also useful in robot control.

Both tiers of the model are robot-centric instead of global. The robot's dead reckoning thus does not need to be very accurate except for small intervals, eliminating the need for error estimation and correction [Cassandra et al., 1996].

When new vision information is received, the corresponding sectors are updated. Sectors that are not in the visual field are updated using odometry, i.e., according to the robot locomotion, enabling the robot to "remember" what it has recently seen. If a sector has not been updated by vision for a time greater than t_{reset}, the range stored in the sector is reset to d_{reset}.

Micro sectors are 5° wide. Due to imperfect image processing, the model

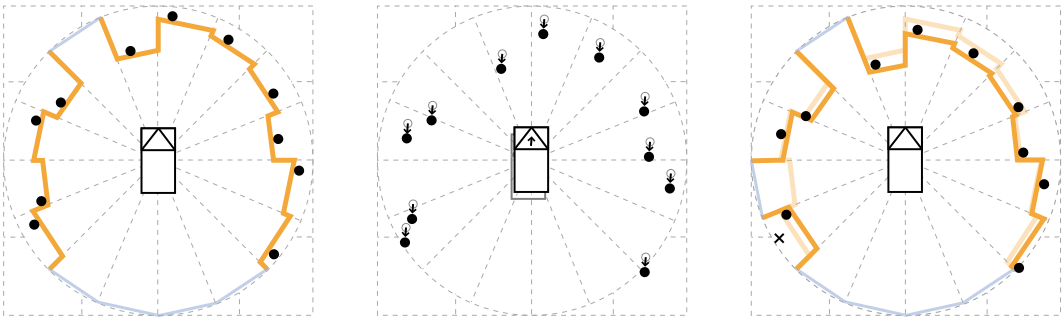


Figure 4.5: Illustration of the two levels of the obstacle model. The solid orange line represents the proximity map, the small dots represent the measurements map. The actual number of sectors is greater than shown here, it was reduced for illustration purposes (Fig. 4.8 for the actual obstacle model). Motion update: in b), the measurement map is moved according to odometry information. This updated information is used to create virtual sensor readings in c) and thus create the new proximity map.

is often patchy, e.g., an obstacle is detected partially and some sectors are updated while others may not receive new information. Instead of using the model as such, analysis functions that compute information from the model are used. These functions produce high level output such as "how much free space is in front of the robot," which is then used by the robot's behavior control layer. These functions typically analyze a number of micro sectors. Depending on the task, the proximity map or the measurements map is used.

The analysis functions typically perform a conservative, worst case analysis on the model, i.e., the single most dangerous obstacle from a micro sector determines the output of the analysis function. Perception and modeling are tuned accordingly to avoid false positives that greatly impair performance. As a matter of fact, bad color calibration results in an increase of false positives and has a very negative effect on performance.

Abstracting from the model by using analysis functions makes using the model robust against errors introduced by inaccurate sensor information. It also offers intuitive methods to access the data stored in the model from the control levels of the robot. The following paragraphs will explain in more detail how the model is updated and cover some useful analysis functions.

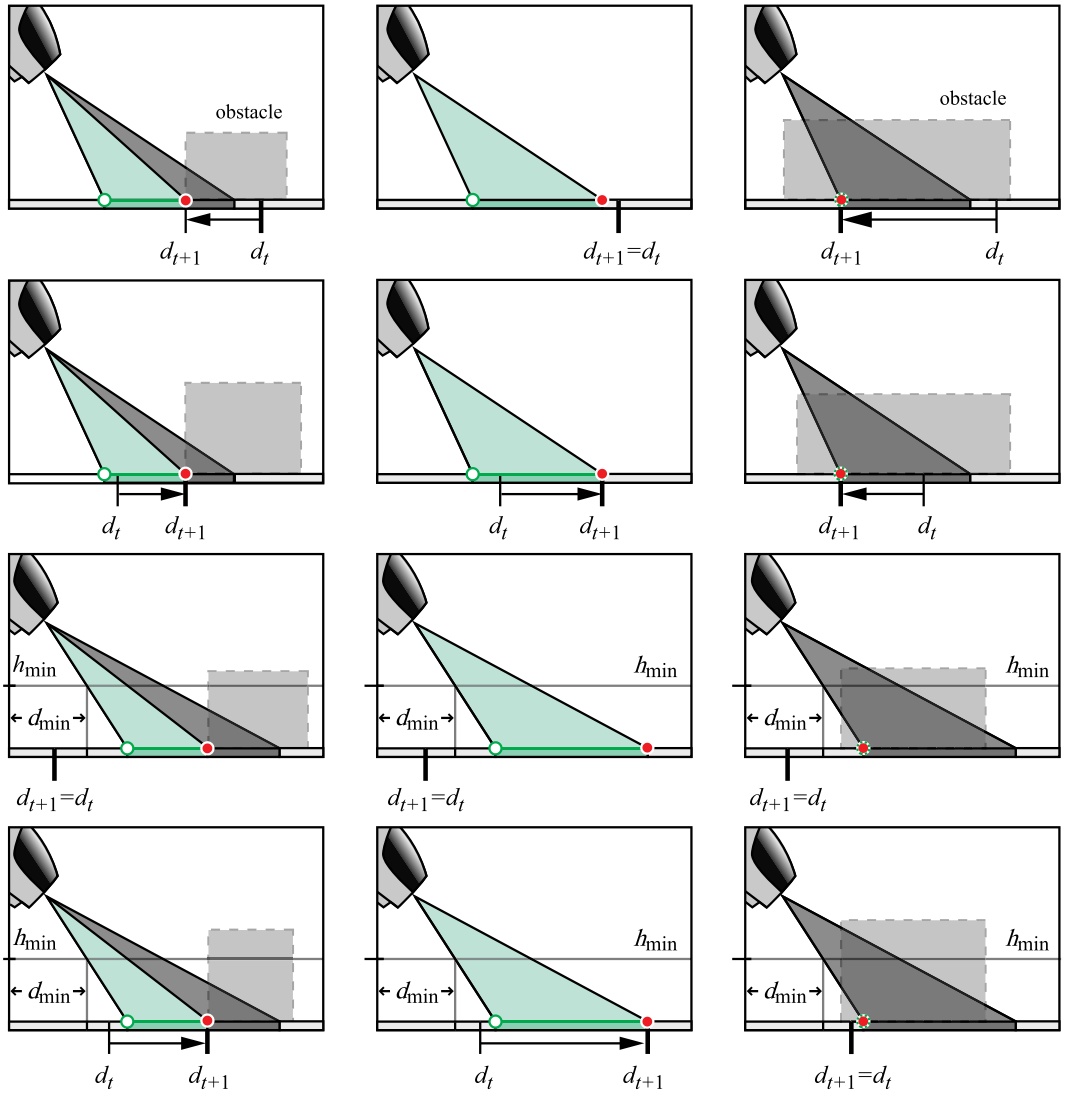


Figure 4.6: Illustration of what can be deduced from the current sensor reading given the distance stored in a sector.

4.2.3 Model Update

Integrating Vision Data

The camera image is analyzed as described in 4.2.1. Obstacle percepts are used to update the obstacle model. The detected free space for each of the vertical scan lines is first associated to the sectors of the obstacle model. Then the percept is compared to the free range stored for a sector. In contrast to actual sonar, the current sensing range, i.e., the area visible to the robot, has

to be considered carefully. Fig. 4.6 illustrates several possible cases that may occur when updating the information stored in a sector θ. Important for this update is whether or not the modeled distance is within the current field of view and also if the near point of the obstacle percept lies on the image border.

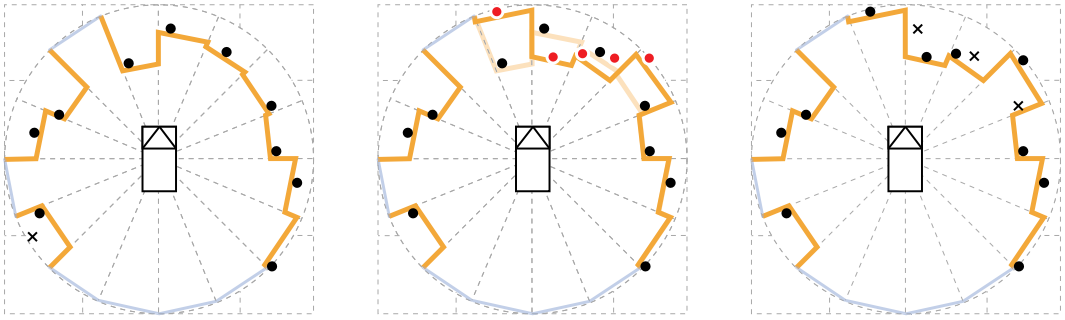
If the distance in a sector was updated using vision information, the measurement map is also updated using the obstacle percept as a representative of that sector. The necessity to store this information will become clear later.

Motion Update

Sectors that are not in the visual field of the robot (or where image processing did not yield usable information) are updated using odometry. The representatives in the measurement map are moved according to the robot's motion. The updated representative is then re-mapped to the – potentially new – sector. It is then treated like an obstacle detected by vision and the proximity map is re-calculated. In case more than one representatives have moved into one sector, the representative closest to the robot is used for calculating the free space while the farther away ones are discarded (see Fig. 4.7b for an example). If a representative has moved out of a sector and no other representative ends up in that sector, the free space of that sector is reset to d_{reset}. The model quality deteriorates when representatives are mapped to the same sector and other sectors are left empty. [Lenser and Veloso, 2003] shows how these gaps can be closed using linear interpolation between formerly adjacent sectors: whenever a gap between two formerly adjacent sectors of the obstacle model occurs, one or more new points are created for the sectors in-between the two by linear interpolation. This eliminates the deterioration effect but makes the assumption that two adjacent obstacle representatives belong to the same obstacle, which is not always the case. While the resulting model "looks nicer," no gain in obstacle avoidance performance was observed, which in part is due to the way the analysis functions abstract from the model.

Another way of looking at the proposed model emphasizes the similarity to the VFH approach. Recall that in this approach, a 2D cartesian certainty grid is maintained and a polar map is generated from it. The representatives in our approach can be thought of as the certainty grid used in VFH. However, where VFH stores a certainty measure based on how often the obstacle was detected, we only store the binary information if the obstacle was present the last time the robot looked in that particular direction. This is possible because the vision is tuned to assure that a detected obstacle is truly there. It is also important since the robot is operating in a dynamic environment where obstacles may frequently pop into view, already being dangerously close to the robot.

The coupling of the two tiers of the model allows us to keep the number



a) $t = t_1$ b) $t = t_2$ c) $t = t_3$

Figure 4.7: Observation update: illustrates how the model is updated using odometry when the robot is moving. Updated representatives are shown as dark circles dots, discarded ones are marked by "x".

of obstacle representatives in the measurement map manageable and offers a convenient concept to integrate new sensor readings.

Virtual Sensor

To make obstacle avoidance more robust against sensor imperfections, the Model-Based Dynamic Window Approach [Fox et al., 1998b] uses a virtual sensor that incorporates information based on the current localization of the robot into the obstacle model (given the robot has a map). In RoboCup games, the environment is relatively simple and the obstacle model as presented so far proved highly reliable. However, RoboCup rules forbid field players to enter their own penalty area. We therefore implemented a virtual sensor that, based on the current robot pose, creates distance measurements to the penalty area. These are integrated into the obstacle model as if they were obstacle detections. For computational reasons and because entering the penalty area is not as bad as running into an actual physical obstacle, the virtual sensor is based on the current robot pose only and does not take into account localization uncertainty. This is a simplified version of [Fox et al., 1998b], which uses the entire sample set representing the localization probability density to calculate a worst case distance measurement.

Integration of the virtual sensor reading at the model level keeps the robot from entering its own penalty area without the need of higher level path planning. The virtual sensor could also be used to integrate information about the whereabouts of team mates, but as they are usually fast moving, the benefit is limited. Furthermore, static obstacles such as the goals and the beacons could also be integrated using the virtual sensor. This was not done because

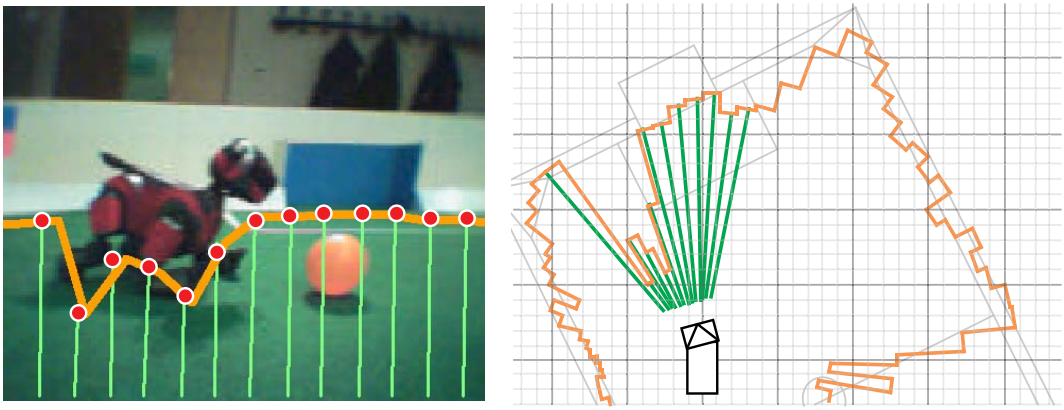


Figure 4.8: *Left:* Camera image with superimposed obstacle percepts and obstacle model (projected onto the floor plane). *Right:* Top down view of the corresponding obstacle model.

it requires a well localized robot and it potentially keeps the robot from going after the ball if it is badly localized.

Model Initialization and Aging

Regions that have not been scanned need to be initialized. The distance stored for the sector is set to d_{reset}. There are two opposing approaches which can be understood as *optimistic* and *pessimistic* views:

Optimistic If a sector has not been scanned, the optimistic view assumes that there is no obstacle in this particular sector. This view is quite plausible in the RoboCup world, where the environment is mostly open space, where there are relatively few obstacles (other robots), and where collisions are not fatal.

Pessimistic In the pessimistic view, areas which have not been scanned are assumed to be occupied by obstacles until evidence of free space is received. This view is useful if the robot acts in more confined spaces, say indoor environments, or if it is in a situation where it is likely to be followed or attacked by another agent (e.g., a robot getting ready to kick the ball).

Similarly, sectors which have not been updated for a certain period are re-initialized and the information is no longer trusted. This aging is done because of the dynamic nature of the environment: other robots move relative to the robot so it is unlikely that an obstacle remains in the same sector at the same distance for a long time. Furthermore, since only odometry is used

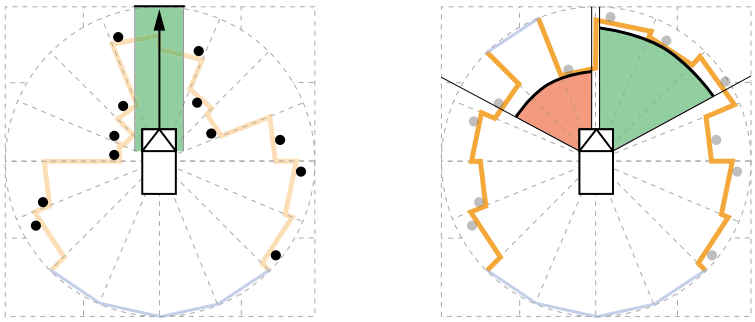


Figure 4.9: Illustration of how analysis functions work. a) Analysis function using the measurements map to determine if a corridor in front of the robot is free to pass. b) To avoid obstacles, the proximity model is used to determine if there is more free space to the left or right of the robot.

to propagate sectors that have not been seen recently, the odometry error increases over time, causing the quality of the obstacle model to deteriorate. Other forms of "aging" are possible, such as a shrinking of the free space to model other robots approaching.

In the experiments below, the optimistic approach was used and aging was implemented by re-initializing sectors after 6 seconds of not being updated.

4.2.4 Accessing the Model

As explained above, the model is accessed by means of *analysis functions*. The micro sectors used to construct the model are of such small dimensions that they are of little use for the robot's behavior control. In regard to the way robot behavior is modeled [Lötzsch et al., 2004], more abstract information is needed, such as "There is an obstacle in the direction I'm moving in at distance d" or "The front left hemisphere offers more free space than the front right." Usually, the obstacle closest to the robot in a given area relative to the robot is of greatest interest. In the following paragraphs, some analysis functions that were used for obstacle avoidance and in RoboCup games are described. Other functions exist for different kinds of applications not covered here.

sect$(\theta, \Delta\theta)$ This function is used to find out how much free space there is in a macro sector in direction θ and of width $\Delta\theta$ using the proximity map. A macro sector is a collection of micro sectors, described by its bearing θ and its angular width $\Delta\theta$. Each sector within the macro sector is analyzed and the function returns the smallest distance in that macro sector. This can be used to construct a simple obstacle avoidance

behavior. The free space in two segments ("front-left," $-22,5° \pm 22,5°$, and "front-right," $+22,5° \pm 22,5°$) is compared and the behavior lets the robot turn in the direction where there is more free space.

corridor(θ, w) It was found that using the proximity map and the above described analysis function, the robot experiences troubles when trying to navigate through narrow openings. For the robot to pass through such an opening, a corridor of free space is needed. corridor(θ, w) returns the distance of the closest obstacle in a corridor of bearing θ and width w based on the measurements map. Typically, a corridor of about twice the width of the robot is considered safe for passing.

free_space_for_turning = **corridor**$(\theta = \pm 90°, \Delta d = l)$ When turning, the robot is in danger of running into obstacles that are to its left or right and thus currently invisible. These areas are checked for obstacles using this function (l = length of robot). If obstacles are found in the model, the turning motion is canceled.

next_free$(\theta, \Delta\theta)$ This function calculates and returns the direction θ_{free} closest to the angle θ where free space of more than $\Delta\theta$ can be found. This is used in situations where obstacles are known to be nearby and helps to determine an angle for the robot to shoot the ball in.

4.3 Obstacle Avoidance Behavior

In this section, a robot behavior that makes use of the obstacle model to avoid collisions is described. Sample applications in the RoboCup domain are described and experimental results are given.

Goal-Directed Obstacle Avoidance as Used in the Challenge

The basic idea of the obstacle avoidance mechanism presented is to have the robot turn towards the goal when obstacles are far. This is combined with a forward motion that adapts to the proximity of obstacles, slowing the robot down when obstacles are near. If the robot finds itself in a situation where it needs to actively avoid obstacles, it clings to the obstacle and tries to circumvent it until it can again advance toward the goal. This is achieved by the following control mechanisms:

a. **Turning towards the target** The robot turns toward the goal if the space in front of it is greater than a threshold value (i.e., no obstacles are in its way).

b. **Controlling the robot's forward speed** The robot's forward speed is linearly proportional to the free space in the corridor in front of the robot.

c. **Turning towards open space** If the free space in the corridor in front of the robot becomes less than a threshold value, the robot will turn towards where there is more free space (i.e., away from obstacles).

d. **Override turning toward goal** If there is an obstacle to the left or right of the robot that it would run into while turning, turning is omitted and the robot will continue to walk straight ahead.

Motor commands are low-pass filtered to produce smoother robot motion. In a typical run, the robot starts moving and turns toward the goal (*a.*). It continues this trajectory until an obstacle appears in the corridor in front of it when (*b.*) causes it to slow down and (*c.*) causes it to turn away from the obstacle in the direction where there is more free space. As soon as it has turned far enough so that no obstacle is in front of it anymore, (*b.*) will result in a speed up. An oscillation between (*a.*) and (*c.*) will result in a direction that allows the robot to pass the obstacle on the tangent to the obstacle. The robot is also able to follow walls, as (*a.*) and (*c.*) result in the robot moving parallel to the wall. As soon as it has passed the obstacle, (*a.*) causes the robot to turn towards the goal. To make sure that the robot has completely passed the obstacle, (*d.*) checks the area beside the robot to make sure that turning motions do not cause the robot to run into obstacles.

Behavior (*b.*) also serves as an emergency stop when obstacles suddenly appear in the robots way and the robot needs more time to turn away from them.

Obstacle Avoidance in RoboCup Games

In the RoboCup championship games, the obstacle avoidance approach was used in conjunction with a force field approach to allow for various control systems to run in parallel. The obstacle model was also used for shot selection: as the robot approaches the ball, the model is used to check if there are obstacles in the intended direction of the shot. If obstacles are in the way, the shot direction is altered accordingly.

Scanning Motion of the Head

The control of the robot's gaze direction is important for obstacle avoidance to work. We will briefly go over some of the used head motions:

Hard Wired Sweeping Motion Obstacle avoidance works best if the robot performs a dedicated scanning motion with its head. This gives the robot effective knowledge about its vicinity (as opposed to just its field of view), allowing it to better decide where it should head. The scanning motion and the obstacle avoidance behavior are fine tuned to allow for a wide scan area while making sure that the area in front of the robot is scanned frequently enough for it to not run into obstacles. The robot's head is also tilted sufficiently upward for it to see landmarks and be able to localize.

Active Vision In the actual RoboCup games, the camera of the robot is required to look at the ball most of the time. Therefore, very little dedicated scanning motions are possible resulting in an inferior model of its surroundings. (Scanning for obstacles was later included in the active vision system.)

Actively Scanning for Obstacles This is achieved by using the obstacle model to drive the head motion. As the head performs a left-right scanning motion similar to the one described above, its tilt is adjusted to direct the camera to where obstacles are expected according to the obstacle model. This ensures that the percepts actually deliver the information needed to update the model (see also Section 4.2.3). This type of head motion has a positive side-effect on localization: if detected obstacles are far away, the robot's gaze will automatically be along the horizon, where landmarks are most likely to be seen. If obstacles are near, the robot lowers its head and scans the areas relevant to updating the obstacles model. It does not miss any landmarks by doing this, as landmarks are likely to be obstructed by the nearby obstacle anyway.

Anticipating Robot Motion Since the current motion of the robot is known and also the target direction, the gaze direction can be adjusted to make sure that sectors the robot is headed in are monitored more closely than others. This is achieved by adjusting the center of the scan and its amplitude.

4.4 Experimental Results

RoboCup 2003 Technical Challenge

The objective of the obstacle avoidance challenge was for a robot to cross the field from one goal to the other as quickly as possible without running into any of the other seven robots placed on the field. The other robots did not move and were placed at the same position for all contestants (Fig. 4.10).

Figure 4.10: RoboCup World Cup 2003 obstacle avoidance challenge: The robot has to cross the entire field from one goal to the other. Seven other, non-moving robots on the field had to be avoided. The image shows our robot entering the penalty area close to finish. The space between the 2 red robots was only about 3 times wider than the robot's width, posing a great difficulty for many participating teams (see Fig. 4.11).

The presented approach fared well against the other contestants. Fig. 4.11 show the results of the teams participating in the challenge. As can be seen from the results, the system enabled the robot to move quickly and safely across the field. Avoidance was highly accurate: on its path, the robot came very close to obstacles (as close as 2 cm to touching the obstacles), but never touched any. Very little time was lost for scanning the environment (as the obstacle model is updated continuously while the robot's head is scanning the surroundings), enabling the robot to move at high speeds without stopping. Please note that this exceptional performance was achieved even though the system was not optimized for static obstacles and also not specifically tuned for speed.

RoboCup 2003 Championship Games

The obstacle model was used for obstacle avoidance and for shot selection in the games. An improvement in game play was noticeable when obstacle avoidance was used. In several instances during the games, situations in which the robot would otherwise have run into an opponent, it was able to steer around it.

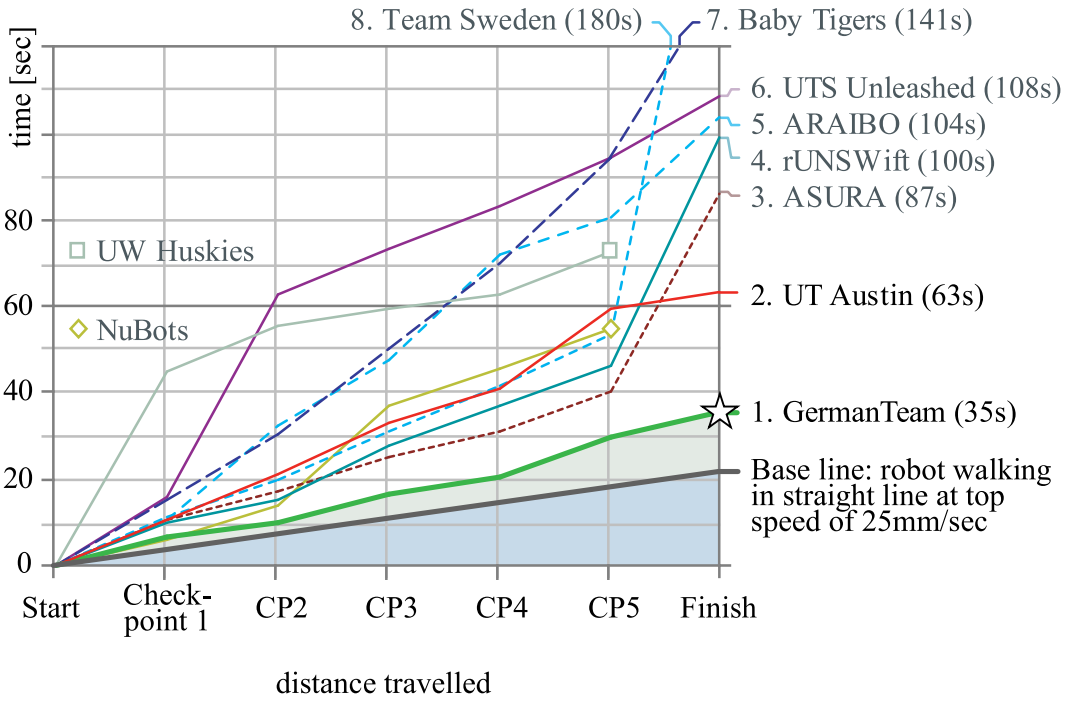


Figure 4.11: Speed of the fastest of the 24 teams participating in the challenge. Teams that did not make it beyond the first obstacle are not shown. Note that many teams had difficulties with the obstacle configuration near the finish (UW Huskies and NuBots where unable to pass the last obstacles and enter the goal).

Collision avoidance is most obvious when robots return to their kick-off positions after a goal has been scored. Here, robots are able to steer clear of other robots and safely and quickly return to their designated positions.

Practical Considerations

Image processing needs to be tuned to produce few false positives, as these have a negative effect on the model. Bad color calibration, for example, can cause the robot to detect obstacles all around it. The model's accuracy is also diminished by the robot's camera shaking when it is walking, resulting in an error in the distance measurements.

Odometry errors cause distortion of the model, but since a robot-centric frame of reference is used and the model is updated frequently, no negative effects on performance are observed.

The control policy works well in experiments and in RoboCup games. It can be lured into situations where it performs poorly (maze like set-ups, dead

ends), but these rarely occur in RoboCup.

One challenge does remain in competitive situations: obstacle avoidance causes the robot to stay clear of obstacles. While this is, of course, the goal of collision avoidance in a static world, it is not necessarily the optimal action in competitive situations, where obstacles are likely to be opponent robots. Here, the detour can cause the robot to arrive at the goal later than the opponent robot. To fully cope with this, the dynamics of the environment need to be considered, ultimately leading to path planning, taking into account (short term) intentions of other robots.

4.5 Summary

Although the field of obstacle avoidance is well studied, very little work has been done in the Sony League due to sensory limitations of the platform. The presented system enables the robot to reliably circumvent obstacles and reach its goal quickly. The system was developed to achieve local obstacle avoidance in dynamic environments using limited field of view vision data. This requires a model of obstacles close to the robot to make sure the robot does not collide with obstacles currently out of sight. The combination of conservative visual obstacle detection, robot-centric two-layered modeling, and analysis functions to access the model proved to be highly robust and successful. The GermanTeam won the RoboCup 2003 obstacle avoidance challenge, reaching the goal almost twice as fast as the runner up without touching any obstacles.

Modeling obstacles in the vicinity of the robot builds the basis of using negative information described later, where information about obstacles is used to determine if landmarks are occluded.

Chapter 5

Proprioceptive Motion Modeling

5.1 Introduction

In the previous chapter, collision avoidance was described. However, due to the dynamic nature of the environment the robot operates in, collisions with other robots occur even when collision avoidance approaches are employed. Collisions are particularly bad for the Sony Aibo as two robots easily get one another's legs entangled, making directed locomotion impossible. On the one hand, the robot needs to recover from this state of impaired mobility as quickly as possible. On the other hand, collisions leave the robot badly localized, since its actions do not have the expected result. The same is true not only for collisions, but also for minor hinderances and imperfections of the robot's locomotion.

This leads back to the original example of the robot running into a wall. The robot ends up in this situations because obstacle avoidance has failed. To make matters worse, it has no way of recognizing this situation as it lacks the necessary means to update its belief or to trigger a recovery action. It therefore remains in a state of confusion, acting erratic, unable to free itself from the situation. This chapter will present remedies to this problem.

Proprioception

In humans and animals, *proprioception* (also referred to as *kinesthesia*) is the perception of stimuli relating to the individuals own position, posture, balance, or internal condition, i.e., the sense of the position of parts of the body relative to other parts in combination with the sense of balance located in the inner ear. Proprioception is of high importance for locomotion and many everyday tasks. It does, for example, allow a person to touch his or her nose with eyes

closed. Proprioception is generated from stimuli originating in muscle spindles (a specialized muscle structure innervated by both sensory and motor neuron axons), in the golgi tendon organs within muscle fibers, in pressure sensors in the skin, and in the vestibular system. The vestibulo-ocular reflex, for example, stabilizes the image on the retina by eye movement in the opposite direction of head movement. In rare cases, patients lack proprioception due to brain damage. They then rely completely on their visual system to achieve body movement and locomotion, making it very hard for them to perform everyday tasks [Sachs, 1985]. Interestingly, most robotic systems are also designed with few sensory/proprioceptive inputs and vision as the primary (and sometimes only) sensor.

The Sony Aibo does not have any dedicated touch sensors on its shell, making it difficult to detect if it is touching another object. Even the Aibo's touch sensors in its paws have difficulty detecting if a robot's leg touches the ground. To cope with this lack of dedicated sensory input and to provide a degree of proprioception, the movement of the robot's legs is examined. The robot's legs (and to some extent its head) are the parts of the robot that come into contact with other robots when collisions occur. It was found that not only the overall movement of the robot, but the movement of individual limbs is impaired during a collision. Monitoring the deviation of intended motions (control) to actual motion of limbs (servo readings), collisions can be detected. Experiments show that for simple motions (e.g., walking straight ahead at constant speed), collisions and obstructions can reliably be detected. As soon as the robot starts pursuing a goal in the dynamic RoboCup environment, such as chasing the ball, the control commands change at high frequency. In order to cope with arbitrary motion commands and accelerations, the system is extended by a layer that monitors collisions over time to avoid false positives. Once collisions are reliably detected, the robot can perform recovery actions to free itself from the other robot.

To understand the effect of collisions on localization, one has to consider the motion update of the Bayes filter employed. In the motion update, the current localization belief is updated using knowledge about the currently intended action of the robot. For a moving robot, this intended action is given by the motor commands and the expected outcome (*odometry*, see also Section 3.3.2). This *motion model* is commonly measured by having the robot move about in its environment and measuring the deviation between desired and actual motion. Collisions are usually not included in the motion model as they are difficult to detect and the effect on the outcome of motions is hard to model. Of the various implementations of the Bayes filters, particle filters offer a great robustness with respect to errors caused by such un-modeled phenomena and are therefore able to preform well on the *kidnapped robot problem*

(see Section 3.1). Most approaches rely on this inherent robustness by putting limited trust in odometry and by assuming a high level motion uncertainty. This approach comes at the cost of accuracy, since the odometry error is artificially increased. Furthermore, the particle distribution cannot reflect the uncertainty brought about by a collision.

For example, the standard localization module used by the GermanTeam does not model collisions at all. This static motion model assumes a worst case error that subsumes everything from unhindered motion to leg slippage to errors that occur by quickly changing robot control commands to collisions with other robots. In contrast, it will be shown how the proposed collision detection approach can be used to create a proprioceptive motion model by explicitly modeling collisions. Not only can this model take into account errors caused by collisions, it can also be used to model odometry errors when the robot experiences small hindrances. This establishes a dynamic level of trust in odometry; uncertainty is increased as collisions occur, resulting in an increase in the entropy of the particle distribution. This particle distribution better resembles the current state of the robot, which allows it to recover more quickly from collisions as particles are more spread out. Entropy can be monitored and used to determine if the robot can go on with its task or if it is better to stop and re-localize before proceeding.

The importance of better modeling arises in part from the low particle count used. With high particle counts, areas of low probability are represented by a small number of particles. If something unexpected happens, such particle filters can recover quickly since unlikely areas are not completely deserted. As the particle count is lowered, however, particle filters become more susceptible to errors caused by un-modeled events as regions of low probability in the state space may not be represented at all. Spreading out the particle distribution when disturbances occur helps cope with such events and yields quicker convergence when new sensor data is acquired.

Outline First, a summary of the concept of probabilistic motion modeling is given and the motion model used for the Sony Aibo is described, followed by a brief overview of sensors and related work. Then, the approach to detect collisions and jerky motions of a legged robot based on proprioception is presented. It is used to enable enable the robot to recover from collisions on a behavior level and also to better model robot motion.

5.1.1 Probabilistic Motion Modeling

The motion model is a probabilistic description of the outcome of control action u_t performed by the robot. It is given by the conditional probability density:

$$p(s_t|u_t, s_{t-1}) \tag{5.1}$$

Here, the state s_t is the robot pose, but it may also include the state of interesting objects in the robot's environment. Control actions may be locomotion, e.g., walk sideways (0 mm/s, 150 mm/s, 0 rad/s), or the robot manipulating its environment, e.g., by kicking the ball. Actions may change the robot's pose and may also change the state of the environment (if the ball has been kicked successfully or the robot has run into it).

There are two types of approaches to modeling the motion of the robot: *velocity motion modeling* and *odometry based motion modeling*. The velocity motion model is based on the predicted outcome of control actions only. In contrast, the odometry based model uses an odometer to count the number of turns of the wheels as the robot moves. Given the diameter of the wheels, the distance traveled can be calculated. Both velocity and odometry based motion models suffer from drift and slippage. However, the odometer constantly *measures* the turns of the wheels, whereas discrepancies of actual motion and model are not compensated in the velocity motion model. As the Aibo has no wheels and deducing the distance traveled from the leg's movement is almost impossible, we used a velocity-based motion model. That said, the term odometry is used to describe the predicted distance traveled.

Information about the environment stored in the map can also be integrated into the motion model. For occupied regions in the map, the a-priori probability of the robot to be in such a region is zero $p(x_t|m) = 0$. Likewise, the probability of a control action to move the robot into an occupied region is also zero.

Motion Model of the Sony Aibo

As mentioned above, the term odometry is commonly used to describe how far the robot has traveled given a sequence of control command. A control command is also called a "motion request" and consists of desired robot speeds $\dot{x}, \dot{y}$ and angular velocity ω:

$$\vec{m} = (\dot{x}, \dot{y}, \dot{\theta} = \omega)^{\mathrm{T}} \tag{5.2}$$

Given the motion request, the robot trajectory lies on a circle of radius $r = \frac{|v|}{\omega}$, where $|v| = \sqrt{\dot{x}^2 + \dot{y}^2}$ is the translational speed of the robot (see Fig. 5.1). Note that this radius becomes infinite when the robot is moving

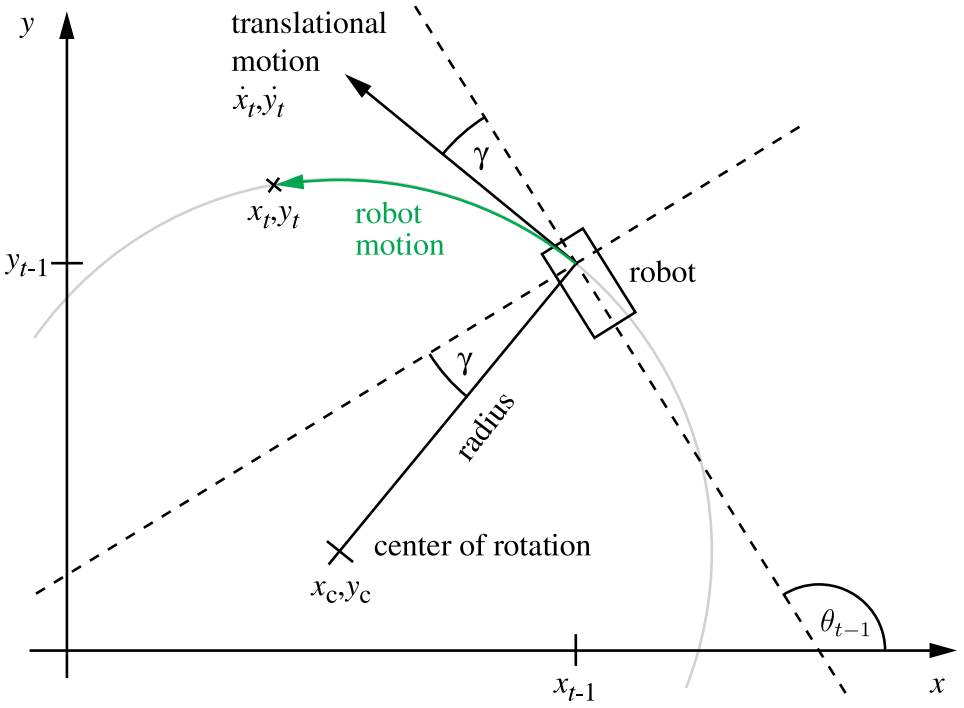


Figure 5.1: Updating the robot pose. The robot moves on a circular trajectory around the center of rotation. The dashed lines indicate the robot coordinate system.

in a straight line and does not turn. The center of the rotation is given by:

$$x_c = x_{t-1} - \frac{|v|}{\omega}\sin(\theta_{t-1} + \gamma) \tag{5.3}$$

$$y_c = y_{t-1}\frac{|v|}{\omega}\cos(\theta_{t-1} + \gamma) \tag{5.4}$$

$$\gamma = \arccos\frac{\dot{y}}{|v|} \tag{5.5}$$

Using the center of rotation, the robot pose at time t can be calculated from the pose at time $t-1$ as follows:

$$x_t = x_c - \frac{|v|}{\omega}\cos(\omega \cdot \Delta t) \tag{5.6}$$

$$y_t = y_c - \frac{|v|}{\omega}\sin(\omega \cdot \Delta t) \tag{5.7}$$

$$\theta_t = \theta_{t-1} + \omega \cdot \Delta t \tag{5.8}$$

Here, Δt is the discrete time step between $t-1$ and t. On the Sony Aibo, $\Delta t = 8\,\text{ms}$ is used.

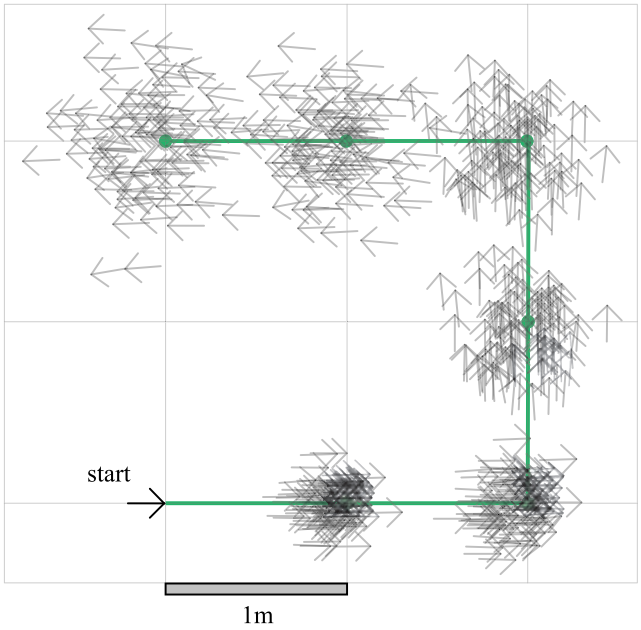


Figure 5.2: Particle representation of the robot's belief as the robot moves. The position uncertainty increases with every step taken; the solid line indicates the localization using based on dead reckoning.

In reality, the distance traveled is never exactly the same in two subsequent runs. Odometry is subject to cumulative errors (drift) and by itself does not account for slipping, sliding, or skidding. We model the odometry error as a normally distributed random variable of finite variance, resulting in the effective speed $\vec{m}_{\text{eff}}$ and effective motion $\Delta\vec{r}_{\text{eff}}$:

$$\vec{m}_{\text{eff}} = \vec{m} + \vec{\epsilon}(\dot{x}, \dot{y}, \omega) = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \omega \end{pmatrix} + \begin{pmatrix} \epsilon_{\dot{x}}(\dot{x}, \dot{y}, \omega) \\ \epsilon_{\dot{y}}(\dot{x}, \dot{y}, \omega) \\ \epsilon_{\omega}(\dot{x}, \dot{y}, \omega) \end{pmatrix} \tag{5.9}$$

$$\Delta\vec{r}_{\text{eff}} = \Delta t \cdot \vec{m}_{\text{eff}} \tag{5.10}$$

The effective velocity of the robot equals the command velocity plus an additive error ϵ of zero mean. Note that ϵ is a function of all three control inputs, i.e., drift may cause the robot to turn although $\omega = 0$. In a simple model, the noise is assumed Gaussian and the standard deviation σ_i of the PDF of ϵ_i is assumed linearly proportional to the weighted sum of the control inputs, requiring a total of 9 parameters to describe the odometry error:

$$\sigma_i(\dot{x}, \dot{y}, \omega) = \alpha_{i,1}\ \dot{x} + \alpha_{i,2}\ \dot{y} + \alpha_{i,3}\ \omega \tag{5.11}$$

When updating the current belief using the motion model, $\vec{m}_{\text{eff}}$ is used in Equations 5.6-5.8. In the case of a particle filter, this means for each particle, an error is sampled from the error PDF associated with ϵ. This error is added to the motion request, which is in turn used to calculate the new robot pose at time t. In the actual MCL particle filter implementation used by the GermanTeam, each particle's pose is updated using the motion request $\vec{m}$. The odometry error $\Delta t \cdot \vec{\epsilon}$ is added after this update. Furthermore, the standard deviation in Eqn. 5.11 is approximated in the following fashion:

$$\sigma_i(\dot{x}, \dot{y}, \omega) \approx \sigma_i(|v|, \omega) \tag{5.12}$$

This is done under the assumption that the effect of the translational error is the same in dimensions x and y. The robot pose of a sample is thus updated by:

$$\vec{r}_t \leftarrow \vec{r}_{t-1} + \Delta t \cdot \vec{m} + \Delta t \begin{pmatrix} \beta_1 |v| \text{ rand}(-1,1) \\ \beta_2 |v| \text{ rand}(-1,1) \\ (\beta_3 |v| + \beta_4 \omega) \text{ rand}(-1,1) \end{pmatrix} \tag{5.13}$$

Where $\beta_1, ..., \beta_4$ are parameters describing the error model and $\text{rand}(-1,1)$ is a function that returns a random number in the range $\{-1,1\}$ (uniform distribution is used for computational speed). Since the time interval is constant, the factor Δt in 5.13 can be included in the parameters:

$$\vec{r}_t \leftarrow \vec{r}_{t-1} + \Delta t \cdot \vec{m} + \begin{pmatrix} \beta'_1 |v| \text{ rand}(-1,1) \\ \beta'_2 |v| \text{ rand}(-1,1) \\ (\beta'_3 |v| + \beta'_4 \omega) \text{ rand}(-1,1) \end{pmatrix} \tag{5.14}$$

The values of parameters $\beta'_i = \Delta t \cdot \beta_i = 8\,\text{ms} \cdot \beta_i$ used in the implementation are:

$$\beta'_1 = 0.1\,\text{ms},\ \beta'_2 = 0.02\,\text{ms},\ \beta'_3 = 0.002\,(\text{rad/m})\text{ms},\ \beta'_4 = 0.2\,\text{ms} \tag{5.15}$$

Inverse Kinematics The previous paragraphs describe how to predict the motion of the robot given the current robot speed $\vec{m}$. For this desired motion to translate into actual locomotion of the robot, the movement of all its joints has to be controlled. In the GermanTeam code, this is the task of the walking engine. It calculates the necessary joint angles at any given time for the robot to move about using *inverse kinematics*. Please refer to [Düffert and Hoffmann, 2006] for details.

Modeling Kicking the Ball

Besides modeling the robot's locomotion, the outcome of the robot interacting with its environment can also be modeled. In the soccer domain, the most im-

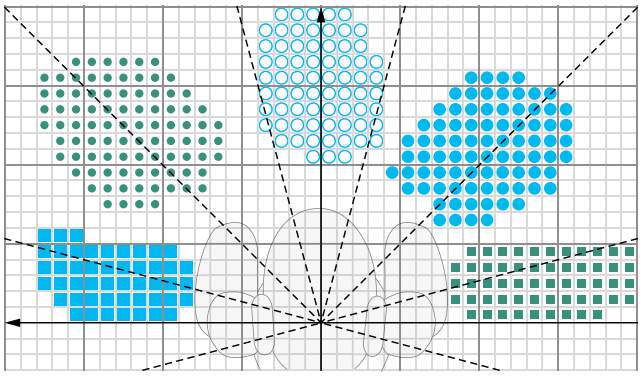


Figure 5.3: Kick selection table for 5 different kicking motions (different markers in the grid).

portant interaction is kicking the ball. The GermanTeam uses a large number of individual kicking motions sot the robot can directly kick the ball within its kick range without having to first align its body to it. For each kicking motion, the ball has to be in a specific area relative to the robot for the kick to be performed as desired. Each motion kicks the ball in a specific direction. Both properties are stored in the *kick selection table*. The table is used to select an appropriate kick for a desired kick direction depending on the current robot pose and game situation. It also allows to probabilistically model the direction of motion of the ball after a kick has been executed.

5.1.2 Related Work

Modeling Proprioception

Many research efforts in mobile robotics aim at enabling the robot to safely and robustly navigate and to move about both known and unknown environments. Please also see Section 4.1.1 for a description of sensors and navigation approaches in the context of obstacle avoidance.

Wheeled robots are widely used in environments where the robot can move on flat, even surfaces such as office environments, environments that are accessible to wheelchairs [Lankenau et al., 2002], but also on less structured surfaces found outdoors or on planetary surfaces [Yoshida et al., 2002]). Using visual, tactile, and vibration-based sensory information, the terrain surrounding a planetary rover can be analyzed and classified, allowing the robot to adapt to its current surroundings [Iagnemma et al., 2004]. Using reinforcement learning, the control of a wheeled, omnidirectional robot can be learned and thus adapt to a surface of given properties [Gloye et al., 2005].

Legged robots are generally believed to be able to deal with a wider range

of environments and surfaces than wheeled robots. The many designs of legged robots vary mainly in the number of legs, ranging from insectoid or arachnoid with 6, 8 or more legs [Clark et al., 2001], 4-legged, such as the Sony Aibo used in our research [Düffert and Hoffmann, 2006; Hornby et al., 2000], to humanoid with 2 legs (biped) [Behnke et al., 2006; P. Dario, 2001].

Apart from the current action failing, collisions (and subsequently being stuck) have severe impact on the robot's localization as odometry is used in the localization process [Dellaert et al., 1999b; Röfer et al., 2005]. As stated before, the Sony Aibo lacks dedicated sensors that would allow the robot to detect touching objects and collisions. This leaves two sensors to achieve proprioception and thus collision detection, the accelerometer and the directional sensors of the robot's servos.

Proprioceptive Sensors

Accelerometer The Sony Aibo features a built in accelerometer. An accelerometer measures acceleration forces, which may be static and constant, as is gravity, or dynamic, as caused by movement or vibration. The acceleration is measured by determining the displacement of a weight due to inertia causing a change in electronic properties of the system, such as capacitance, piezoelectric effect, etc. The Aibo's accelerometer returns a real valued measurement of the robot's acceleration in space $(\ddot{x}, \ddot{y}, \ddot{z})$. The values are limited to the range of $[-2\,\mathrm{g}, +2\,\mathrm{g}]$, with $\mathrm{g} = 9.81\,\mathrm{m/s^2}$, and are sampled at 125 Hz. We found the accelerometer data of the Aibo to be very noisy and it was only used in situations where low pass filtering could be applied. One such application is to have the robot's head look parallel to the ground based on the gravitation vector. The most important application, though, is to use the accelerometer to determine if the robot has fallen over, resulting in the gravitation vector to point up (within the robot's frame of reference), and then trigger a recovery action. In preliminary work [Vail and Veloso, 2004], Vail and Veloso propose to use the accelerometer for a number of further applications: training a surface detector on accelerometer data, estimating robot velocity, and detecting if the robot is free, entangled, or stuck. The key idea is similar to the one proposed by Quinlan et al., where reference measurements are used to classify the current situation [Quinlan et al., 2004] (see below). This work does not, however, deliver evidence that the proposed applications can actually be implemented using the accelerometer data.

Servo Motors A servo motor is an electrical motor with an integrated position feedback device and controller. The position is measured by what is called the *resolver/encoder*. The measured position is fed into a controller that tries to match the desired position. Commonly, a PID-controller is utilized.

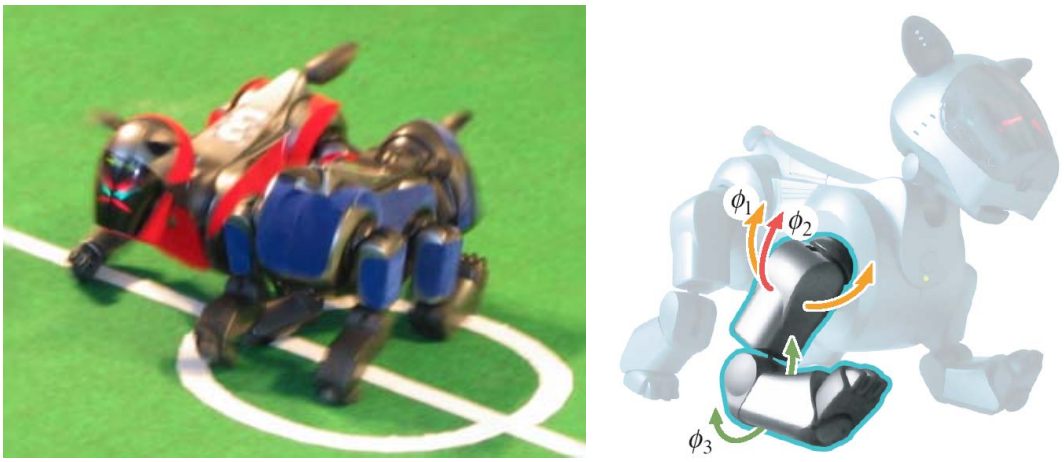
PID-control considers the deviation of desired and actual position (*proportional*), the deviation integrated over time (*integral*), and its rate of change (*differential*). The weighted sum of the proportional, integral, and differential component yields the control signal.

The Aibo uses servo motors in all of its joints. The position/direction measurements of these servos can be used to estimate the robot's posture. Quinlan et al. [Quinlan et al., 2004] show how these measurements can be used to effectively monitor the traction of the robot, namely by comparing the current servo direction measurements of the leg joints to reference values gathered prior to the run. It relies solely on the direction measurements and does not take into account the control commands. The reference data consists of the average sensor data value and variance for a given motion type for a discrete time grid over the period of one step. This training is done by measuring the sensor data of possible combinations of elementary motions. A four-dimensional lookup table is used to store the reference data. The four dimensions of the table are: forward/backward motion ("back stride length"), sideward motion ("strafe"), rotation ("turn"), and a time parameter, which stores information about the relative position of the paw in its periodic trajectory. Using this approach, the "Nubots" were able to detect collisions and slip. However, the four-dimensional lookup table requires a considerable amount of memory and training time; in their work, they used 20x12x20x20 entries to fully describe a gait with each entry consisting of two floating point values (average value and standard deviation). During training, it is important that no collisions or slip occur, requiring external supervision.

Using this lookup-table, no assumptions are made about similarities between actuator command and sensor readings. In contrast, the approach presented here is based on the assumption that there is, in fact, a similarity between intended and actual motion and that the variance of the sensor signal is bounded for the entire period of the motion. Under these assumptions, training can be greatly simplified as the number of parameters required to describe unhindered motion can be reduced by several orders of magnitude. The reference value is proportional to the control signal, which is calculated by the walking engine as the inverse kinematic calculations are performed [Düffert and Hoffmann, 2006].

Motion Model

While improving the sensing model has been the focus of many papers (see 3.6 and also bibliographical remarks in [Thrun et al., 2005]), modeling the motion has received little attention and quite crude models prevail in the context of robot localization [Röfer et al., 2005; Thrun et al., 2005]. In [Kwok and Fox, 2005], the motion of a ball is described by states, taking into account



a) b)

Figure 5.4: a) A collision of two robots. Neither robot can move into the desired direction. Even worse, robots often interlock their legs, which further prevent them from resolving the situation. b) Illustration of the degrees of freedom (DOF) of the Aibo. Each robot leg has three joints, two DOF in the *shoulder joint* and one DOF in the *knee joint*, denoted ϕ_1, ϕ_2 and ϕ_3. Joints are labeled in the following fashion: F(ront) or H(ind) + L(eft) or R(ight) + Number of joint $(1, 2, 3)$. Using this nomenclature, the knee joint of the highlighted leg in the above image is *FR3*.

interactions of the ball with its environment. Such interactions cause the ball to transition from one state to another. Each transition is modeled statistically, where the probability of state transition is a function of the robot's current localization, its current belief regarding other robots, and its own actions. Our approach takes this idea and applies is to modeling robot locomotion, using proprioception rather than belief as evidence for what state the robot might find itself in.

5.2 Collision Detection

In this section, the approach to collision detection using the Sony Aibo is presented. The approach is threefold, (1) being based on comparing control commands and actual motion, (2) requiring the two signals to be aligned and (3) requiring some filtering of the control commands to account for real life situations.

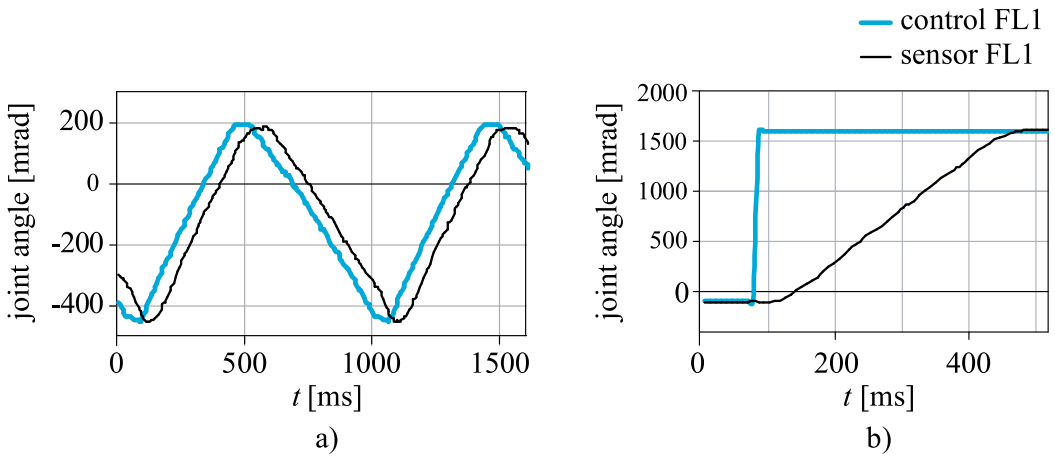


Figure 5.5: a) Sensor and actuator data of freely moving legs (in the air) at a desired ground speed of 75 mm/s. Sensor and actuator curves are almost congruent except for a slight phase shift. b) Sensor and actuator data for a rectangular actuator impulse. The actuator function jumps to its new value. The corresponding servo's direction sensor readings are shown.

5.2.1 Sum of Squared Deviations

The approach is based on the comparison of control (actuator) commands to direction sensor readings of the Aibo's servos. Fig. 5.5 shows typical sensor measurements alongside the control signals. It can be seen that for a period T of unhindered movement, the sensor and actuator curve of a joint are congruent, i.e., they are of the same shape but shifted by a phase $\Delta\varphi$. When the curves are aligned, the area in between the two curves becomes minimal:

$$0 \leq \int_{t_0}^{t_0+T} (a(t) - s(t + \Delta\varphi))^2 \, \mathrm{d}t \tag{5.16}$$

In our experiments, discrete time intervals of length 8 ms are used (this is also called a *frame*). Experiments show that collisions cause a discrepancy between actuator and sensor data, which can be recognized by calculating the area between the sensor and actuator data. It was found that it was not necessary to sum over one complete period of the motion to detect collisions. Shorter intervals yield faster response times. Trading off response time and sensitivity to sensor noise, we found that 12 frames were sufficient. The last 12 frames thus are used to calculate the sum of squared deviations (similar to the sum of squared errors used in least squares function fitting):

$$S_{a,s}(\Delta\varphi) = \sum_{i=t_1}^{t_2} (a_i - s_{(i+\Delta\varphi)})^2 \tag{5.17}$$

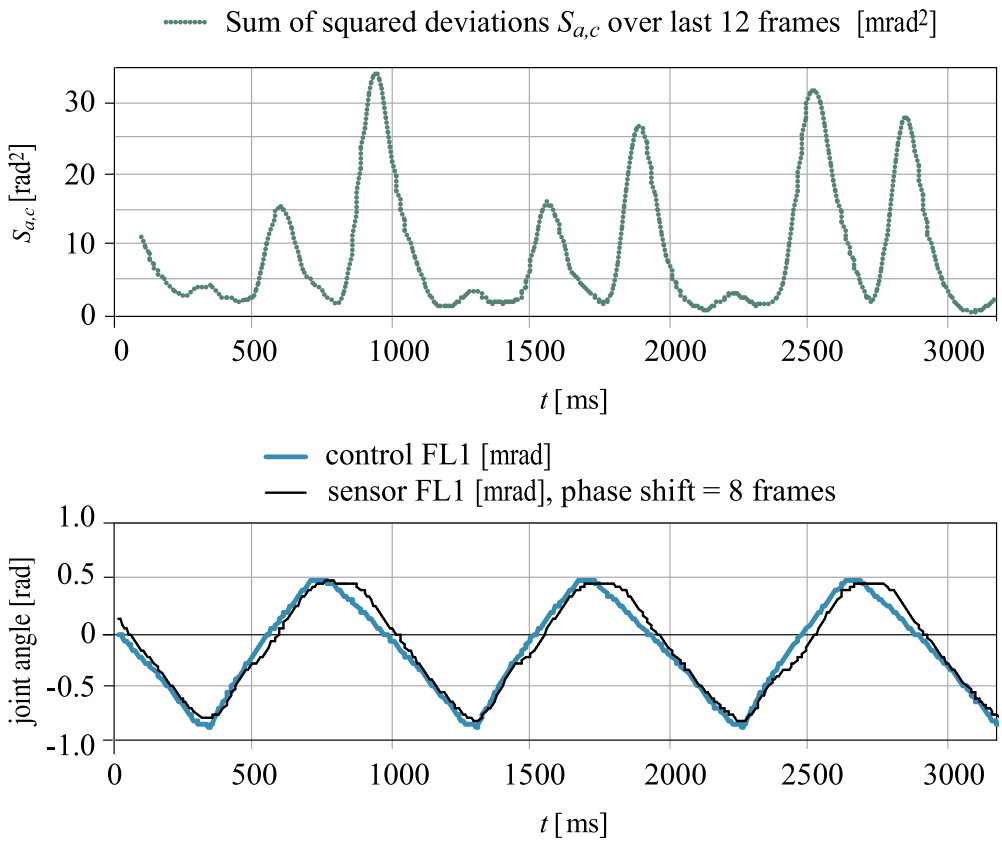


Figure 5.6: Sensor and actuator data of a collision with the field boundary walking forward at 150 mm/s (i.e., the robot is running into a wall). In $S_{\mathrm{a,c}}$, the collisions (a leg hitting the wall) can be seen as peaks in the curve. The peaks occur briefly after the actual collision and can easily be distinguished from unhindered movements.

Diagram 5.6 shows $S_{a,s}$ of the *FL1* joint (left shoulder) for a robot colliding with the field boundary. Peaks in $S_{a,s}$ clearly correlate to the robot's leg hitting the field boundary. (In this diagram, the curves are aligned, which is described in the following section.)

For classification of collisions, $S_{a,s}$ is compared to a threshold. If $S_{a,s}$ is larger than this threshold, it is assumed that a collision has occurred. The thresholds for every motion component (i.e., walking forward/backward, walking sideways, rotation) are stored in a lookup table. For combined motions (e.g., walking forward and walking sideways at the same time), the different thresholds for each motion component are summed as described in Section 5.2.5.

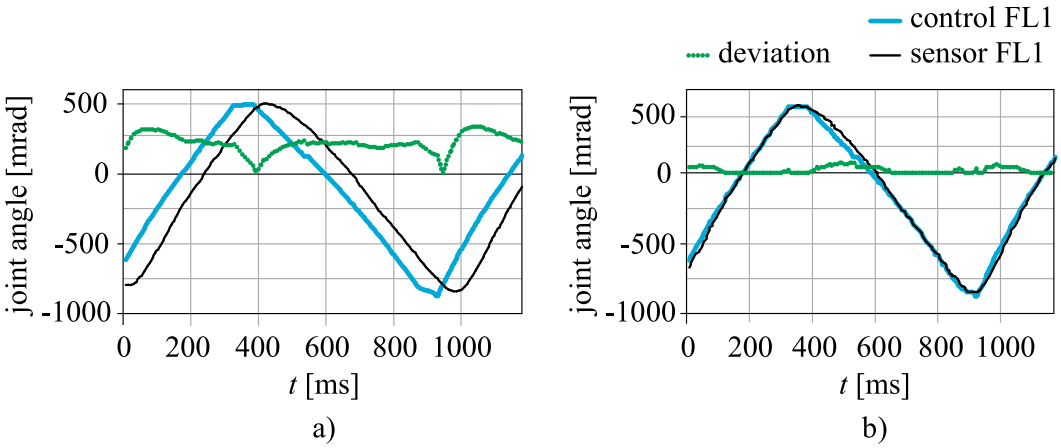


Figure 5.7: a) Sensor and actuator data for walking freely at 150 mm/s. Actuator and sensor curve are out of phase and the corresponding sum of squared deviations $S_{\mathrm{a,c}}$ is shown. b) As above but phase shifted. Sensor function is shifted by 8 frames. The corresponding $S_{\mathrm{a,c}}$ now clearly shows collisions (peaks in the curve).

5.2.2 Aligning Actuator and Sensor Curve

Fig. 5.5 shows the impulse response of one of the robot's servo motors. It can be seen that the joint exhibits a latency of about 5 frames (40 ms). The latency is due to joint momentum and also by the time the system needs to process the request; furthermore, the step height and the load that the joints have to work against have an influence on the observed phase difference. After 5 frames, the joint slowly starts moving and accelerates until it reaches its maximum speed after 8 frames. Just before reaching its destination, the speed decreases due to the joint's PID controller smoothing the robot's motions.

In Figure 5.7(a) $S_{a,s}$ is shown for a sample motion. The smallest values of the $S_{a,s}$ are found at the intersection of the two curves. Collision effects have little influence on the difference level. In Figure 5.7(b), actuator and sensor curves are aligned by shifting the sensor data curve 8 frames to the left. The calculated $S_{a,s}$ shows a strong response to collisions.

In experiments, phase shifts of varying lengths can be observed. Since no reason for the variations in length was found, the curves are aligned such that $S_{a,s}$ becomes minimal.

$$S_{\min} = \min_{\Delta\varphi} S_{a,c}(\Delta\varphi) \tag{5.18}$$

This approach eliminates phase shifts not caused by collisions and reduces the risk of wrongly recognizing collisions (false positives). Since the range of

possible $\Delta\varphi$ is limited, real collisions still produce a strong signal.

5.2.3 Filtering of Actuator Input

The so far presented approach to collision detection works well under laboratory conditions, i.e., when applied to homogeneous motions with small, well defined motion changes. In real world applications, motion commands may change rapidly as the robot interacts with the environment. In the dynamic, highly competitive RoboCup domain, the robot changes its walking speed and direction at high frequency as determined by the behavior layer of the agent. Figure 5.8 shows the actuator commands for a robot playing soccer. Most of these changes are relatively small and do not pose a problem for the approach, but some are too extreme to be executed by the servos, e.g., when the robot suddenly sees the ball and moves towards it at the highest possible speed. This is compensated by increasing $S_{a,s}$ threshold if the joint acceleration exceeds a certain value. This increased threshold is used only for fractions of a second and then falls back to its initial level.

5.2.4 Threshold Calibration

The values of the thresholds are calibrated manually. They are measured for each of the elementary motions (forward/backward, sideways, rotation) in steps of 30 mm/s and 0.5 rad respectively. This adds up to a total of 40 measurements needed for operation.

A threshold is determined by having the robot walk freely and without collision or slip on the field for about three seconds while monitoring both motor commands and sensor readings. S is calculated and the maximum $S_{\max}$ is used to derive a threshold value. The maximum value $S_{\max}$ that occurred is tripled and used as the threshold value, i.e., for the robot to detect a collision, the current S must be 3 times greater than the maximum $S_{\max}$ encountered during calibration.

In our experiments, the calibration was done by hand, since robot gaits do not undergo frequent change and the calibration process is performed quickly. There is no specific need for automating the calibration process (given that an external supervisor has to make sure that no collisions occur during calibration anyway).

The underlying assumption of the similarity of control and measurement is not always valid. While the walking engine used by the GermanTeam in 2004 is based on the wheel model and uses trapezoidal trajectories for the robot's paws [Düffert and Hoffmann, 2006], the walking engine of 2005 used much more complex trajectories. Furthermore, the genetic optimization employed to find the gait pattern exploited the properties of the PID-controller, creating paw

trajectories that have little in common with those calculated using forward kinematics given the control commands of the joints [Dahm et al., 2005]. For such gait patterns, approaches based on the sensor readings alone are more flexible yet require laborious calibration [Quinlan et al., 2004].

5.2.5 Recognizability of Collisions

For different walking directions, collisions have different effects on the robot's joints depending on how the joints are hindered in their motion. Therefore, the following cases were investigated. In our experiments, only the legs' servos were used for collision detection. However, the robot's head motors can also be used to directly detect whether a robot hits an obstacle with its head (or its head's freedom of motion is otherwise impaired by an obstacle).

In the detection of collisions, a trade off has to be made between being sensitive to collisions and being robust against false positives (the detection of a collision where in reality the robot was moving freely). To avoid false positives, the threshold value for detecting collisions is raised at the cost of being less sensitive to detecting collisions. Furthermore, by integrating the information gathered over a short period of time, false positives can be suppressed. This, on other hand, makes the approach less reactive.

Elementary Motions

Elementary motions are the basic, individual modes of locomotion available to the robot. Arbitrary motions are generated by superposition of these.

Walking Forward or Backward Collisions are easily detected in the front left or right shoulder joints *FL1* and *FR1* of the robot, depending on which of the legs hits the obstacle (see 5.4). This works well for collisions with the field boundary or other static objects. Collisions with other robots can also be detected, but not as reliably because this sort of collision is much more complex (the other robot may also be moving, the shape of the obstacle, etc.). Collisions when walking backwards are slightly harder to recognize because of the particular position of the joints of the hind legs. This is due to the robot's body being tilted forward and the backward motion not being symmetric to the forward motion.
The rate of detection of collisions during forward movement was about 90%; for backward movement, it was about 70%. Sometimes collisions are not detected because the robot pushes itself away from obstacles rather than being hindered in its joints' motions. No false positives were observed due to our choice of the threshold.

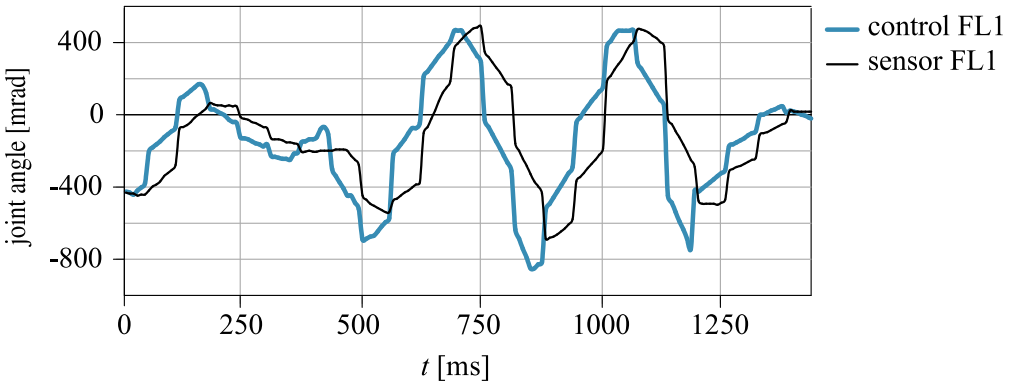


Figure 5.8: Actuator commands and sensor measurements during an actual RoboCup game. The robot is changing directions frequently. It can be seen that the servo is unable to perform the requested motions.

Walking sideways Collisions which occur while the robot is walking sideways can be recognized best in the sideways shoulder joint θ_2 close to the obstacle (e.g., *FL2*). This is not quite as reliable as in the case of forward motions because the Aibo loses traction more quickly when walking sideways for the gait that was used. About 70% percent of the actual collisions were detected. Some phantom collisions were detected at a rate of about 1-2 per minute in typical game situations.

Turning The same joints that are used to recognize collisions while moving sideways can also be used to recognize collisions when the robot is turning. This way, a common type of collision can also be detected: The legs of two robots attempting to turn interlock and prevent the rotation from being performed successfully. How well this can be recognized depends on how much grip the robots have and on the individual turning (or moving) speeds.

The detection rate of collisions and the rate of false positives is about the same compared to when the robot is moving sideways. When raising the detection threshold to completely eliminate false positives for a robot rotating at 1.5 rad/s, the detection rate drops to about 50%.

Leg Lock

The aforementioned "leg lock" also occurs in situations where two robots are close to each other, e.g., when chasing the ball. Leg lock is detected in the same way collisions are. Therefore, it but cannot be distinguished from other disturbances.

Superposition of Elementary Motions

While it is easy for the robot to recognize the above mentioned motions separately, it is harder to recognize collisions when motions are combined, e.g., when the robot walks forward *and* sideways at the same time. For lower speeds, the resulting motions can be viewed as a superposition of the three elementary motions and the resulting threshold is approximated by the sum of the three individual thresholds:

$$T(\dot{x}, \dot{y}, \omega) = T_{\dot{x}} + T_{\dot{y}} + T_{\omega} \tag{5.19}$$

For high speeds, the requested motions exceed the servos performance. To compensate for this, the collision thresholds are increased using a scale factor $f(v)$:

$$f(|v|) = \begin{cases} 1 & \text{if } v < 50\,\text{mm/s} \\ |v|/100 & \text{otherwise} \end{cases} \tag{5.20}$$

$$|v| = \sqrt{\dot{x}^2 + \dot{y}^2} \tag{5.21}$$

$$T'(\dot{x}, \dot{y}, \omega) = f(|v|) \cdot T(\dot{x}, \dot{y}, \omega) \tag{5.22}$$

With this extension, the method can be applied to practically any kind of robot motion and speed commonly observed in RoboCup games. Accuracy could be further increased by calibrating combinations of elementary motions separately and then interpolating between these, but the accuracy is sufficient for our application.

5.3 Recovery from Collision

Sample Application A simple behavior was implemented in the XABSL behavior mark up language [Lötzsch et al., 2004]: The robot walks straight ahead; if it touches an obstacle with one of its front legs, it stops and turns left or right depending on the leg the collision was detected with. It thus turns away from where the collision occurred and then continues to walk straight ahead again. Fig. 5.9 shows the XABSL state machine used.

This simple behavior works reliably on the RoboCup field, regardless of the types of collisions encountered, e.g., collisions with static obstacles, the field border, or other robots. Collisions are detected with high accuracy, producing less than 10% false positives and a detection rate greater than $> 90\%$. In some rare cases, collisions are not detected immediately because of slippage of the robot's legs. In these cases, the robot recognizes collisions with a small delay

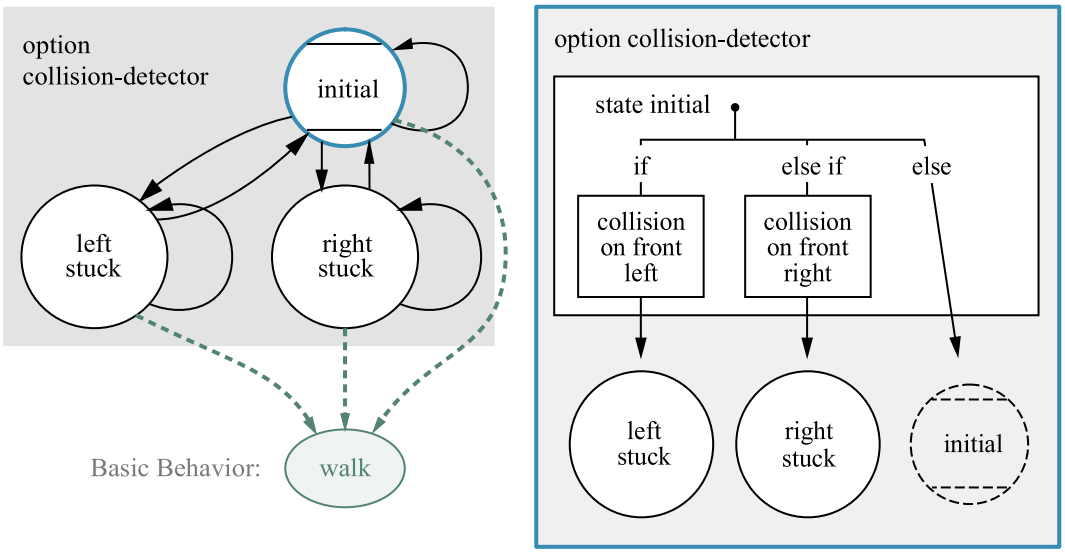


Figure 5.9: State machine describing the collision avoidance behavior. The robot walks forward until it hits an obstacle. It then turns away from it and continues walking in the new direction.

of the order of tenths of seconds (depending on the length of one period of the walking engine).

RoboCup As pointed out by Quinlan et al. [Quinlan et al., 2004], collision detection can be used to make the robot "realize" that an intended action was not successful and to have it act accordingly. However, it proves a difficult task to find the right action in a situation when two robots run into each other. This usually happens when they pursue the same goal, e.g., when both are chasing the ball. Backing off gives the opponent robot an advantage, pushing it makes the situation worse.

One surprisingly efficient way of resolving collisions does not rely on actually detecting collisions: backing off when the ball is no longer detected. This heuristic was originally part of the ball searching behavior (as the robot has difficulties seeing a ball that is right between its paws), but also saved the GermanTeam many "robot pushing" penalties. What happens is the following: As the robot approaches the ball, the ball is visible. If another robot arrives at the ball and both robots run into each other, the ball becomes hard to see for both of them. If the opponent robot manages to gain control over the ball, the ball becomes invisible to (and outside of reach of) the attacking robot. It therefore doesn't make sense to continue to charge for the ball and the robot starts to back off, resolving the collision situation.

5.4 Proprioceptive Motion Model

In this section, I will first describe a simple approach to integrate information about collisions into the motion model as presented in [Hoffmann et al., 2006b]. This naïve motion model is later extended by better modeling collisions and slip. The underlying idea of both approaches is to separate the components of the error brought about by the inherent odometry error $\vec{\epsilon}_{\text{odo}}$ and the error caused by slippage and collisions. This allows to lower the locomotion error per step when the robot is moving freely (accurate odometry) and to only increase this error and thus the belief uncertainty if collisions are detected.

Splitting up the error in Eqn. 5.9 into an odometry component $\vec{\epsilon}_{\text{odo}}$ and a collision $\vec{\epsilon}_{\text{col}}$ component yields the effective speed of the robot:

$$\vec{m}_{\text{eff}} = \vec{m} + \vec{\epsilon}_{\text{odo}} + \vec{\epsilon}_{\text{col}} \tag{5.23}$$

The robot pose is given by:

$$\vec{r}_t = \vec{r}_{t-1} + \Delta\vec{r} + \vec{\epsilon}_{\text{odo}}\Delta t + \vec{\epsilon}_{\text{col}}\Delta t \tag{5.24}$$

$$= \vec{r}_{t-1} + \Delta\vec{r} + \vec{\epsilon}_{\text{odo}}^{\,\prime} + \vec{\epsilon}^{\,\prime}_{\text{col}}{}^{\prime} \tag{5.25}$$

This separation can be used to improve localization accuracy when the robot moves about freely while at the same time enabling it to more quickly recognize collision events. The development was helped by the advent of means to better calibrate the walking engine [Düffert and Hoffmann, 2006], making odometry much more precise as long as movement is unhindered.

5.4.1 Discrete Collision Detection

In a first naïve approach, the robot's motion is monitored and the level of uncertainty in the particle distribution is increased whenever collisions are detected [Hoffmann et al., 2006b]. If no collisions are detected, $\vec{\epsilon}_{\text{col}}^{\,\prime}$ in Eqn. 5.25 is set to zero. If collisions are detected, $\vec{\epsilon}_{\text{col}}^{\,\prime}$ becomes non-zero:

$$\vec{\epsilon}^{\,\prime}_{\text{col}} = \begin{pmatrix} \beta_1 \times \text{rand}(-1,1) \\ \beta_2 \times \text{rand}(-1,1) \\ \beta_3 \times \text{rand}(-1,1) \end{pmatrix} \tag{5.26}$$

In our experiments, the parameters were set to $\beta_1 = \beta_2 = 40\,\text{mm}$, $\beta_3 = 0.5\,\text{rad}$. These values were chosen to model any kind of collision that may occur in our experiments. Note also that in contrast to the odometry error, the collision error used in this model is constant, i.e., it is not a function of the robot's speed.

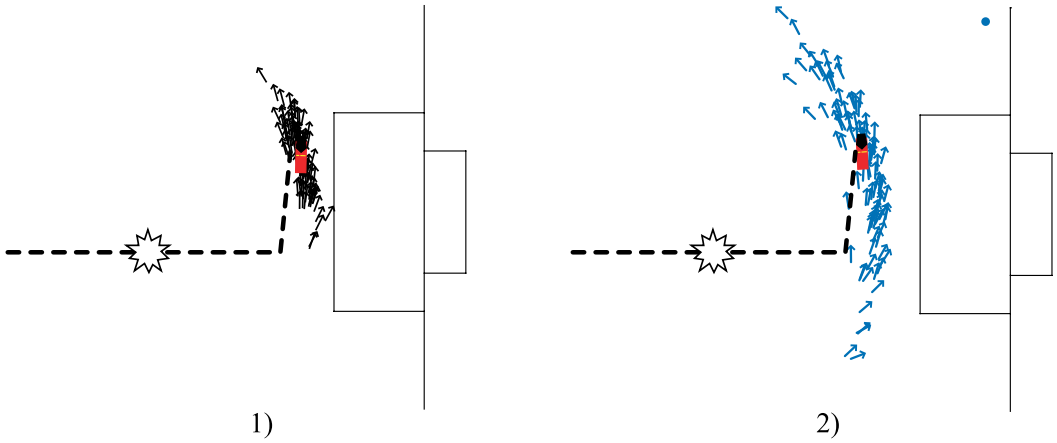


Figure 5.10: Belief distribution when a collision occurs, 1) without and 2) with collision information integrated into the robot's belief. In 2), the uncertainty after the collision is greater and the particle distribution is more spread out.

The result on the robot's belief is best illustrated in an experiment: the robot walks forward from the center circle toward a goal and experiences a collision. It starts out well localized and relies on its motion model only during the experiment, no visual percept data is integrated into its belief. When the collision occurs, the robot is slowed down but not stopped. Afterwards, the robot is able to continue its journey and then turns left. Fig. 5.10 shows the effect of the described motion model on the particle distribution, resulting in a more spread out particle distribution after collisions.

Fig. 5.11 shows the impact on the belief entropy (as defined in Eqn. 3.36) associated with the robot's belief. When collisions are modeled, uncertainty in the belief is clearly visible as an increase in entropy whenever collisions are detected, which can be used to trigger appropriate robot behavior.

5.4.2 States of Mobility

While the discrete approach is a step in the right direction, it has some shortcomings: it only models collisions crudely and it is not able to model the robot being stuck. In this case, the robot experiences an initial collision and is subsequently unable to move. While being stuck, it experiences further collision detections. The simple model is insufficient for this frequently occurring situation. I will show how collisions and the robot's ability to move can be modeled in more detail.

Depending on what situation the robots is in, its ability to move differs greatly. When running into an obstacle, it may not be able to move forward at

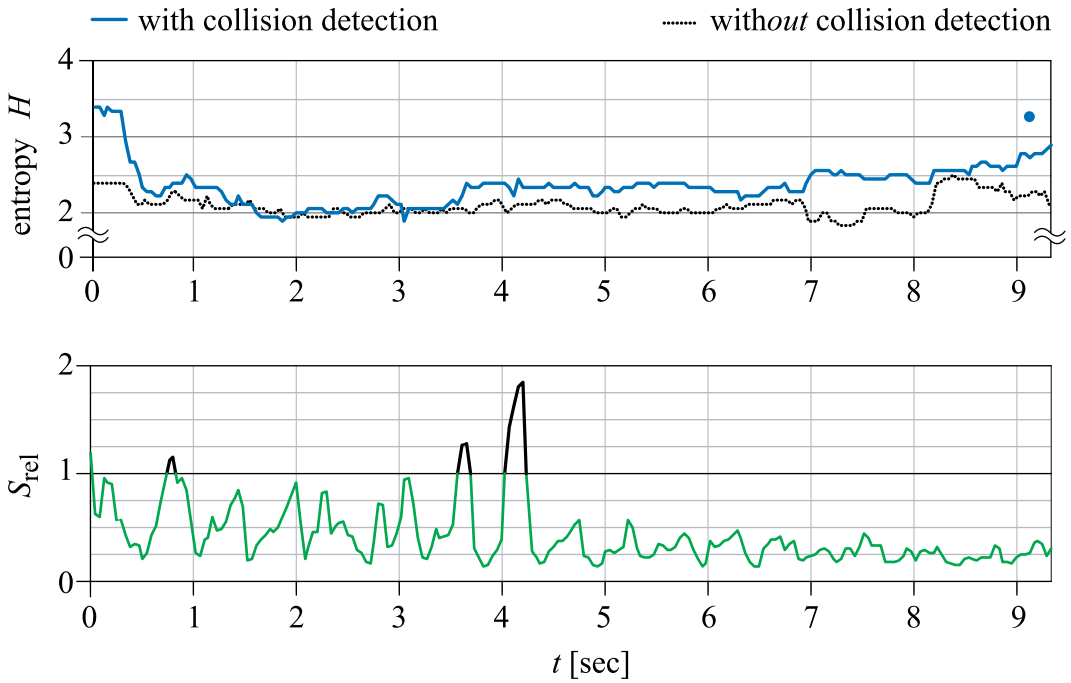


Figure 5.11: Belief entropy when collisions occur: a robot starts walking from the center circle in the direction of the goal and turns left before reaching the penalty area. The distributions entropy with ($\star$) and without the extended motion model (*top*). *Bottom:* Corresponding collision "sensor." The relative sum of squared differences normalized using the collision threshold is shown, values of S_{rel} greater than 1 represent collisions.

all, it may also stumble and fall over, etc. Fig. 5.12 explores the various *states of mobility* of the robot and transitions between these states. The possible states and their effect on the robot's position and orientation are:

Free motion The robot moves freely, no internal or external disturbances occur, odometry error is small.

Slippage This state subsumes motion disturbances like slipping and skidding that occur without external disturbance and that are often caused by abrupt changes of the motion request.
The effective translational speed of the robot is reduced and the robot orientation is subject to error:

$$v' = v \ \text{rand}(\beta_1, 1.0) \tag{5.27}$$
$$\omega' = \omega + \beta_2 \ \text{rand}(-1.0, +1.0) \tag{5.28}$$

with $0 \leq \beta_1 < 1$ and angle β_2.

External hindrance The robot's motion cannot be executed as intended, e.g., because it is running into a wall. The effective motion is smaller than the motion request.
Effective translational speed and angular velocity are reduced:

$$v' = v \text{ rand}(\beta_3, 1.0) \tag{5.29}$$
$$\omega' = \omega \text{ rand}(\beta_4, 1.0) \tag{5.30}$$

with $0 \leq \beta_3 < 1$ and $0 \leq \beta_4 < 1$.

Pushed The robot is being pushed by another robot; a force acts upon it resulting in the robot being turned and displaced.

$$\alpha = \text{rand}(-\pi, +\pi) \tag{5.31}$$
$$d = \beta_5 \text{ rand}(0, 1) \tag{5.32}$$
$$x' = x + d \cos(\alpha) \tag{5.33}$$
$$y' = y + d \sin(\alpha) \tag{5.34}$$
$$\theta' = \theta + \beta_6 \text{ rand}(-1.0, +1.0) \tag{5.35}$$

with displacement error β_5 and angle β_6.

Stopped The robot runs into an obstacle and is stopped dead in its track. The translational speed of the robot becomes zero. However, it is often observed that robots also turn when being stuck:

$$v' = 0 \tag{5.36}$$
$$\omega' = \beta_7 \text{ rand}(-1.0, +1.0) \tag{5.37}$$

with angle β_7.

Penalized A robot that is penalized is placed outside the field by the referee for a certain period of time (most fouls result in a 30 s penalty). The robot is allowed to re-enter the field from either end of the middle line, facing the middle circle. This state is not modeled in the following experiments, but could easily be integrated into the motion model.

Turned Over A robot that has stumbled and fallen on its back will try to get back on its feet by executing a recovery motion. The orientation of the robot after having recovered can be assumed randomly distributed. This state is not modeled in the following experiments, since the experiments focus on proprioception based on servo measurements. The motion model could easily be extended to accommodate this state.

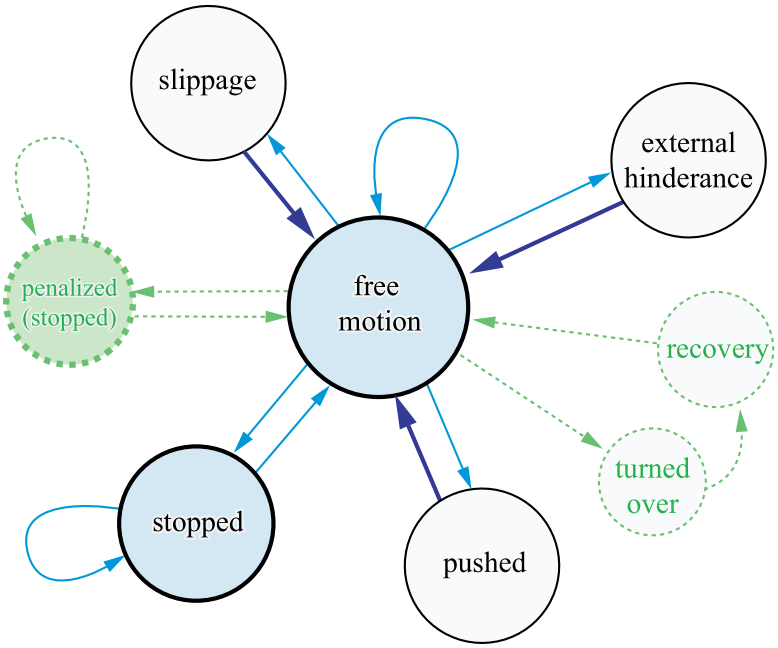


Figure 5.12: States of mobility of the robot: the robot is either able to move freely, or it experiences small disturbances (from which it directly transitions back to "free"), or it is stopped by running into an obstacle, in which case it remains in the "stopped" state for some time. It may also be turned over (e.g., after having been pushed by another robot) and it may also be placed outside the field for 30 s by the referee due to a penalty.

Variables used above: translational speed of the robot $|v| = \sqrt{\dot{x}^2 + \dot{y}^2}$, robot orientation θ, angular velocity $\omega = \dot{\theta}$, and motion model parameters β_i; rand(a, b) is a function that returns a random value in the interval $[a, b]$.

When the robot is in the "stopped" state, it remains in this state for some time until it has freed itself by a recovery action or the entanglement with another robot has somehow been resolved. Characteristically, when the robot is stopped, it will continuously bump into the obstacle, unable to free itself for a few moments. Only when it has freed itself will it no longer detect collisions. It therefore remains in the stopped state for as long as collisions are detected at high frequency.

The remaining collision sates are modeled as transiting directly back to free motion. When the robot bumps into an opponent, collisions are detected in quick succession, sometimes resulting in oscillations between states. This has no negative impact on the motion model, in fact, it reflects the robot being able to move freely for brief periods of time and then being hindered in its motion again.

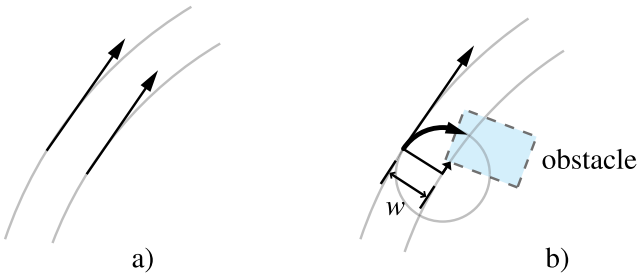


Figure 5.13: a) Vectors representing the speed of the robot's left and right legs. The resulting locomotion is much like that of a differential drive, wheeled robot. b) When the robot's motion is hindered on one side, the forward motion of the unhindered side makes the robot turn.

When collisions occur, the robot tends to turn towards the cause of the collision, usually making matters worse. The reason for this turning is illustrated in Fig. 5.13: the motion of the robot's legs on the side where the collision occurs is hindered, slowing them down. This results in a difference in the forward component of the motion of the left and right legs, causing the robot to turn like a differential drive vehicle. The figure also shows that the maximum turning speed caused by the difference in speeds is given by $\omega = v_{\text{left}}/w$ (with lateral distance between the robot left and right legs w). The collision percept z_{col} contains information about the location of a collision. The angular velocity of the robot thus is changed in Eqn. 5.37 in the following fashion:

$$\omega' = \frac{v}{w}\ \text{turn}(z_{\text{col}})\ \beta_i\ \text{rand}(0, +1.0) \tag{5.38}$$

with:

$$\text{turn}(z_{\text{col}}) = \begin{cases} +1 & \text{if collision occurs on the left} \\ -1 & \text{if collision occurs on the right} \\ 0 & \text{if collision(s) occur left and right} \end{cases} \tag{5.39}$$

This enables modeling of a robot running into a wall, turning towards it until facing it. When finally facing it, it will detect collisions both left and right, marking the end of the turning motion. (Some timing issues need to be considered, as left and right collisions do not take place at precisely the same time, but rather occur alternately.)

5.5 Experimental Results

In the following experiments, a robot moves forward at constant speed and experiences one or more collisions. The belief represented by the particle distribution is generated without external perception, i.e., it is solely based on

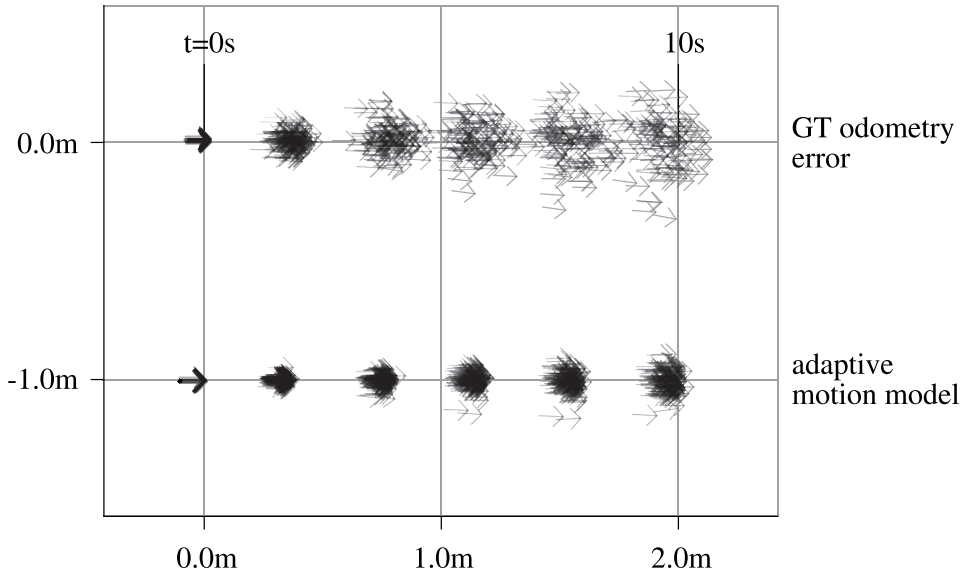


Figure 5.14: Particle distribution representing a robot moving forward at constant speed experiencing no collisions. The *top* distribution uses the basic motion model used by the GermanTeam, the *bottom* distribution uses the proprioceptive motion model. Since the proprioceptive model "knows" that the robot is moving freely, a smaller odometry error can be used compared to the standard model, which has to also model potential external disturbances.

odometry, odometry error, and proprioception-based collision error. This is done to emphasize the effect of the proposed motion model. In an actual application, where vision percepts are constantly integrated into the robot's belief, the proprioceptive motion model makes sure that the particle distribution is spread out enough to model the belief's uncertainty and to allow quicker re-localization after collision events.

5.5.1 No Collision

For comparison, Fig. 5.14 shows the particle distributions of a robot moving forward without experiencing any collisions. Snapshots of the particle distribution are shown in two second intervals. The robot moves at a speed of 200 mm/s and it thus takes the robot ten seconds to cross the distance of two meters. In this illustration and in the following ones, the particle distributions consist of 100 particles. The initial particle distribution has a standard deviation of zero, all particles represent the same robot pose (x_0, y_0, θ_0).

Fig. 5.14 contrasts the particle distribution using the standard GermanTeam motion model and the distribution using the proprioceptive motion

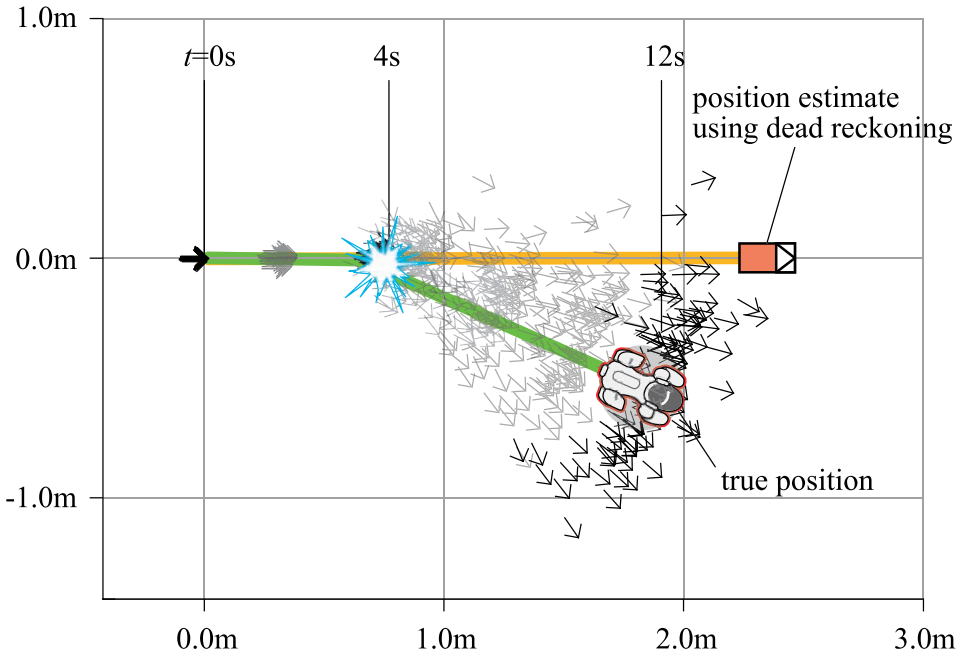


Figure 5.15: Robot running into an obstacle and turning in the process; it then continues to move freely. Note how the particle distribution has become much more spread out compared to the unhindered movement illustrated in the previous figure (5.14).

model. The latter is able to benefit from knowing that the robot is moving freely, using a lower odometry error resulting in a more confined, lower entropy distribution. Since the standard motion model is a function of only the robot speed, a trade off between accuracy and robustness is made: the particle distribution needs to spread out as collisions and hindrances may occur, but at the same time the noise added needs to be limited for the distribution to remain stable and to not diverge too much. The standard model therefore has a higher entropy than the proprioceptive motion model.

5.5.2 Single Brief Collision

In the first collision experiment, illustrated in Fig. 5.14, a robot moves forward at $|v| = 200\,\mathrm{mm/s}$, runs into another robot for about 2 s and then continues to walk forward. The particle distribution of the robot integrating collision information is of typical sickle shape caused by angular disturbances; it is also more spread out after the collision and accounts for the various potential turning motions experienced by the robot. Information about the side on which the collision occurred is integrated and results in a non-symmetric PDF.

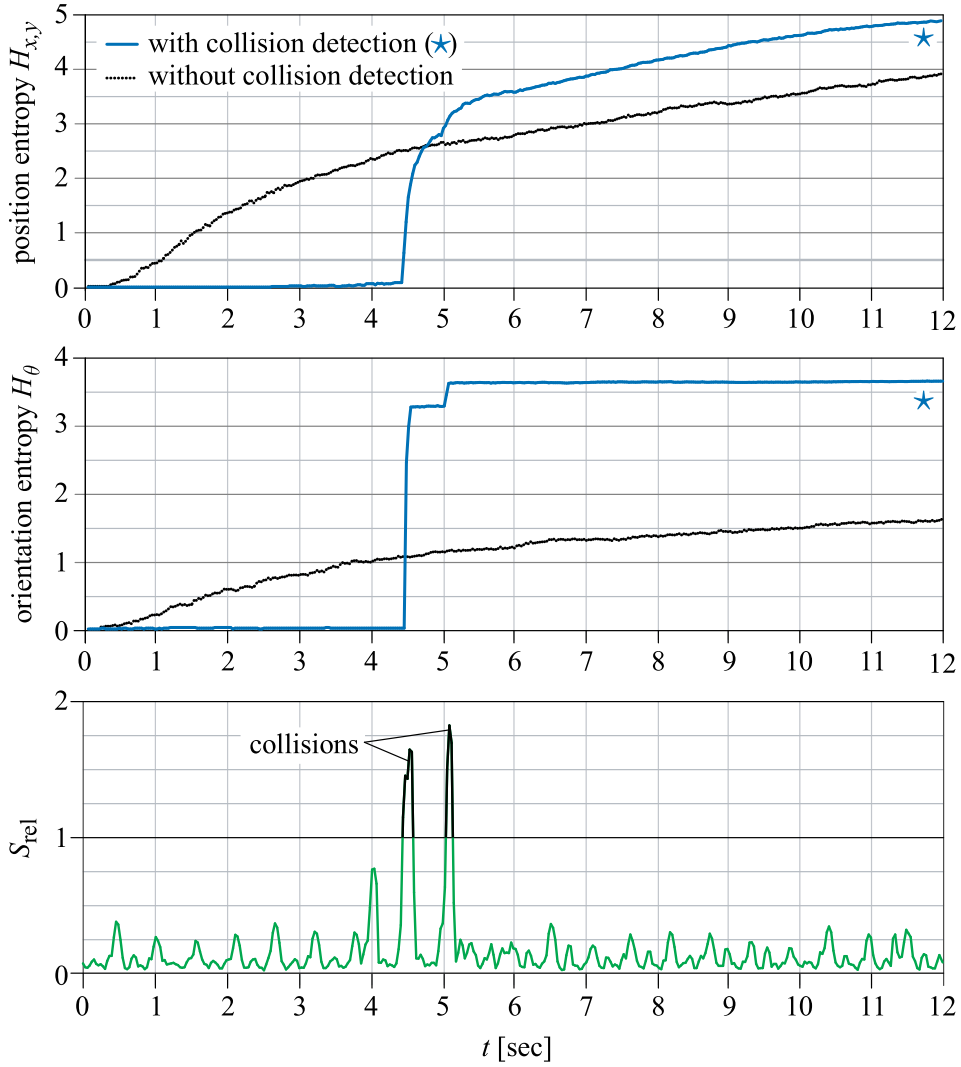


Figure 5.16: Impact of collision on the entropy of the particle distribution. As in Fig. 5.11, values of S_{rel} greater than 1 are interpreted as collisions. The marked curves (⋆) represent the entropy of the particle distribution using the proprioceptive motion model. Without using collision information, the entropy continuously rises over time. Incorporating collision information, the entropy is lower *before* the collision and increases drastically as collisions are detected.

Not taking into account collisions and only using odometry only (dead reckoning), the robot believes to a) have traveled in a straight line and b) to have traveled farther than it actually has.

As intended, the uncertainty associated with the belief increases after the collision. Fig. 5.16 (bottom) shows the relative sum of squared deviations dur-

ing the run, indicating when collisions are detected. The entropy is separated into orientation and position entropy. To generate smoother entropy curves, the area of grid cells was decreased, the grid was centered around the center of gravity of the distribution, and the particle count was increased to 1000 (see Section 3.7). As an artifact of the grid cell size, the entropy values start at zero and remain zero in the beginning of each run. This is because at the very beginning, all particles fall within the center cell, resulting in $H = 1 \log_2 1 = 0$.

Before the collision, the entropy of the proprioceptive motion model is lower than that of the standard motion model. When the collision occurs, the amount of motion noise increases, which can be seen in both the position and the orientation entropy. After the collision, the position entropy of the proprioceptive model continues to rise in a similar fashion as the standard motion model. This is caused by the particles diverging due to the angular uncertainty in conjunction with the robot moving forward.

Please note: The exact shape of the curve is determined by the motion model parameters. The parameters are chosen such that the resulting curve shows the properties described above, i.e., the entropy being small when no collisions occur and increasing when collisions are detected. Fig. 5.16 compares two MCL implementations, each with its own set of parameters defined by their respective designers (see also paragraph 5.6 on calibration).

5.5.3 Two Subsequent Collisions

In a second experiment, the robot moves forward, but experiences two collisions in brief succession, one on its left and one on its right side (Fig. 5.17). The two collisions somewhat compensate each other in terms of resulting robot orientation. However, the robot is slowed down by the collisions and the total distance traveled is smaller than it would have been had the robot been moving freely.

The particle distribution is spread out considerably, accounting for the two collisions and their probable outcomes, including the reduced distance traveled. The impact on the particle distribution of the second collision is not as prominent as the distribution is already quite disturbed. The reduction in distance traveled is quite apparent.

Note that the distribution has become patchy in some areas and not all areas of the PDF are equally well described by the particle distribution due to the small number of particles used.

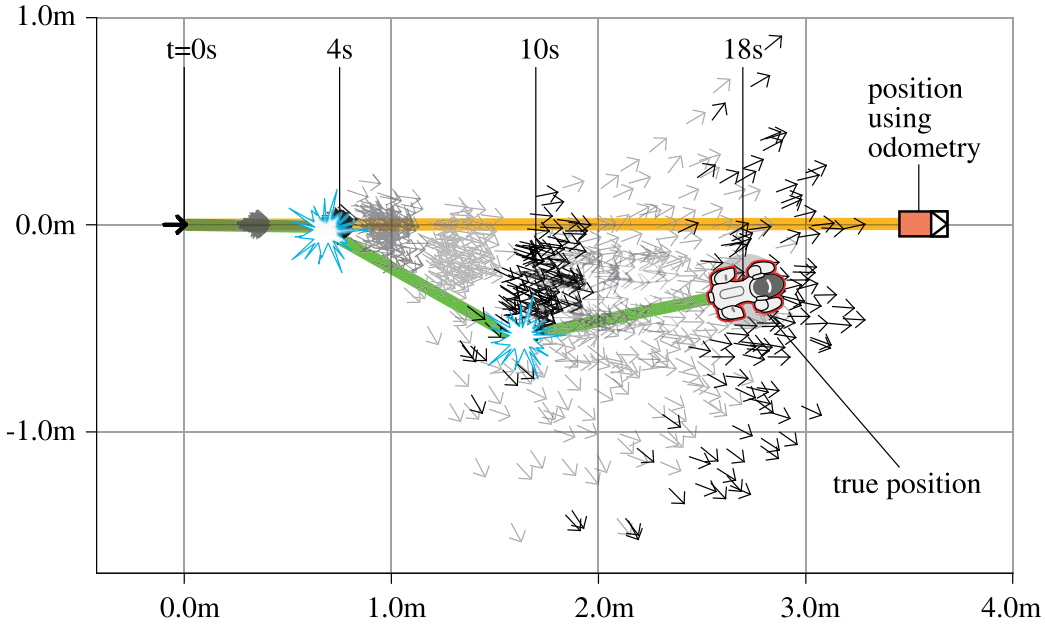


Figure 5.17: Robot experiencing two collisions in brief succession. Note that the distance the robot has actually traveled is much shorter than the distance calculated using only odometry.

5.6 Calibrating the Motion Model

Finding the parameters of the simple, odometry-based motion model is done by calibrating the motion engine. This is achieved by comparing the actual motion of the robot to the current control signal and storing the deviation. Knowing the deviation, the robot control can be adjusted to match actual and intended speed. For the robot itself, it is difficult to determine its own speed on a basic RoboCup field since beacons are not always visible. We therefore used a black and white pattern that allows the robot to localize precisely and update the belief at 33 Hz [Düffert and Hoffmann, 2006]; [Röfer, 2005] uses the Aibo's touch sensors on its feet to detect the ground phase of motions and calculate the robot's speed from this; [Dahm et al., 2005] uses external monitoring in the form of a ceiling mounted camera to measure ground truth.

If the true outcome of the motion is known, the standard deviation is measured in repeated experiments. The error is assumed to be linear with speed. For a robot following a rectangular path using the GermanTeam 2004 walking engine [Düffert and Hoffmann, 2006], the position error after traveling $4 \times 1.5\,\text{m} = 6.0\,\text{m}$ (including four 90° turns) at 200 mm/s is less than $\Delta x = 0.2\,\text{m}$ and the orientation error is less than $\Delta\theta = 10°$. Comparing this to the noise used in the standard motion model in Fig. 5.14, it is obvious that a large

amount of noise is added to account for unforeseen events.

To achieve collision detection, thresholds for elementary motions (forward and sideways translation, rotation) need to be determined. This can be done in separate runs. For threshold calibration, each run needs to be free of collisions.

Systematically determining the parameters needed to describe the proprioceptive motion model that incorporates collision information is much more difficult. Not only is it necessary to detect collisions, but the outcome of collisions must also be monitored. The following aspects need to be considered:

- The effect of a collision depends on the obstacle hit: running into a wall causes a different disturbance than being entangled with another robot.
- Recognizability of collisions depends on the current motion of the robot, e.g., if its legs are touching the ground at the moment of impact.
- Small differences in the way the robot collides can slow down the robot, stop it entirely, and/or cause it to turn.

Addressing this to the fullest, external observation is indispensable to map internal perception to the externally monitored outcome of collisions. Using such rich data, machine learning approaches could be employed to learn the model parameters. A very elaborate model could potentially be created in this fashion, though generating statistically sufficient training data remains challenging. Also, it is not clear if this data holds significant information to justify the effort. I therefore chose to manually adjust the parameters based on a set of sample runs. The resulting model may not be as accurate, but it still retains the single most important property: the entropy of the distribution qualitatively reflects the true localization uncertainty. While in theory, the goal of modeling the motion may be to maximize accuracy, in practice the exact shape of the distribution is not as important [Thrun et al., 2005]. Particle filters are robust as long as the uncertainty is sufficiently modeled, i.e., areas of low probability are represented by particles. With the proposed model, this is guaranteed as the particle distribution is spread out whenever disturbances are detected.

5.7 Summary

With the presented method for proprioception, a 4-legged robot is able to reliably detect collisions with obstacles in its environment. Comparing the requested motor command to the measured direction of the servo motors of the robot's legs provides proprioceptive sensing and proves an efficient method of detecting if the robot's freedom of motion is impaired. A robot behavior is

shown that allows the robot to turn away from an obstacle after having "run into it." Marginally extending this approach, it can be made robust enough to be used in RoboCup games, where motion commands change quickly, which sometimes violates the underlying assumption of similarity between control and sensor data. Collisions with other robots and landmarks on the RoboCup field can be detected and this knowledge can be used to trigger recovery actions.

As collisions also have a negative effect on robot localization, the odometry-based motion model was enhanced using proprioception. In this proprioceptive motion model, states of mobility are considered, allowing the model to scale the amount of odometry noise based on the current situation. This allows to reduce noise in the case of free motion and to increase noise in the case of collisions. Compared to the standard motion model, the resulting entropy of the particle distribution more closely models the current uncertainty of the robot's belief. It allows the robot to move for longer periods of time without having to look at landmarks as long as no collisions are detected. On the other hand, if the robot is competing with another robot for the ball, collisions occur and the particle distribution becomes more spread out. Such a distribution allows quicker re-localization when landmarks are detected and can also be used to trigger actions when the uncertainty becomes too large, e.g., triggering a scanning head motion to validate the localization before taking the next action to make sure that the ball is shot in the right direction.

Chapter 6

Negative Evidence Modeling

6.1 Introduction

In this chapter, the notion of *negative information* is introduced and its power is illustrated in a number of experiments. Negative information is gathered in situations where a sensor reading is expected, but the sensor and subsequent data processing fails to detect a feature. This discrepancy can tell the agent that its current belief is likely to be wrong.

In feature-based measurement models, feature detections are commonly the only information used to update the agent's belief. Failing to detect an object that is in plain sight is called a *false negative* and can have many reasons: sensor imperfections, physical properties of the object being tracked, insufficient lighting, occlusions, etc. Sensors and models are often tuned in a way that *false positives* are avoided at the cost of a reduced detection rate. In the case of the camera used in the Aibo, reasons for not being able to detect an object fall into two categories:

- Extraction of features in image processing of the camera image fails due to problematic lighting, imperfect color calibration, motion blur caused by the robot's head moving too fast, limited camera resolution, and others.
- Objects in the environment can be occluded partially or fully by other robots and are therefore hard to detect or even undetectable by the camera.

A probabilistic sensor model will be proposed which considers both: a) sensing imperfections brought about by the camera, its motion, and image processing, and b) occlusions in the robot's environment. Explicitly modeling occlusions is necessary to ascertain the absence of a sensor reading, in other words to ensure that an undetected feature is truly not there.

While localization is often improved by adding more and more features detected in the raw data (e.g., by using field lines), the novelty of incorporating negative information stems from using the complementary non-detection event of a feature.

Outline First, probabilistic sensor modeling is investigated more closely, as it is the basis of integrating negative information. Then the notion of negative information is introduced and illustrated in two thought experiments. With this in mind, a sensor model for the camera of the Sony Aibo is developed. The beneficial effect of integrating negative information in various situations is then shown in experiments.

6.1.1 Probabilistic Sensor Modeling

Let us recall the two equations describing the iterative version of the Bayes Filter under the Markov assumption (Section 3.12):

$$\text{bel}^{-}(s_t) \longleftarrow \int p(s_t|s_{t-1}, u_{t-1})\text{bel}(s_{t-1})ds_{t-1} \tag{6.1}$$

$$\text{bel}(s_t) \longleftarrow \eta p(z_t|s_t)\text{bel}^{-}(s_t) \tag{6.2}$$

Equation 6.1 shows the *a priori* belief $\text{bel}^{-}(s_t)$, which propagates the previous belief using the motion model of the robot. In Equation 6.2, the probabilistic sensor model $p(z_t|s_t)$ is used to calculate the posterior from the prior probability distribution. The sensor model alongside the motion model (Section 5.1.1) are domain specific models used in probabilistic robotics. The sensor model describes how sensor measurements are generated in the physical world. Modeling the sensor requires an understanding of the physical workings of the sensor and the environment the sensor is used in. Cameras are modeled using projective geometry (pinhole model), sonar requires modeling sound waves being emitted by the robot and reflected back to it. As with all physical measurement instruments, measurements exhibit errors, which is mostly assumed to be Gaussian for convenience. In addition to measurement errors, there are numerous other effects resulting in non-deterministic or erroneous sensor readings: errors introduced by the robot that the sensor is attached to, failure to measure anything (e.g., physical property is out of the sensing range of the sensor), unexpected measurements caused by moving objects in the environment, collisions, etc. The sensor model $p(z_t|s_t) \equiv p(Z = z_t|s_t, m)$ describes the conditional probability distribution of the random variable Z (observation) to have value z_t given the robot is at position s_t and conditioned on a map m. A map is a list of objects and locations $m = m_1, ..., m_N$ describing the environment the robot operates in. Such a map can be feature-based,

as in the case of the landmarks used in RoboCup, or it can be location-based, containing information about objects and free space (occupancy grid map). If the sensor model is known, it can be used to deduce the robot's localization, which can also be seen in the general formulation of Bayes rule:

$$p(s_t|z_t) = \frac{p(z_t|s_t)p(s_t)}{p(z_t)} \tag{6.3}$$

As an example for sensor modeling, let us first consider proximity sensor described in Section 4.1.1 [Thrun et al., 2005]. Such a sensor is subject to measurement uncertainty caused by physical properties of the sensor and the environment: specular reflections may cause range readings that are much larger than the distance to the actual object and cross talk between sensors may result in altogether random distance readings.

Range sensors can be modeled as sensing the range to an object along a beam (laser) or within a cone (sonar, Section 4.1.1). Let z_t' be the true range to the object from the sensor given the current position and orientation of the agent using the map of the environment. The following example is set in the context of localization where a static map is assumed. The maximum sensing range of the sensor is given by $z_{\max}$. The probability of measuring z_t is modeled by taking into account the following sources of error:

Gaussian noise, $p_{\mathbf{error}}(z_t|s_t, m)$ In physical instruments and sensors, the actual measurement differs from the expected reading z_t' by an error Δz_t. This error is mostly treated as a Gaussian distributed, zero-mean random variable with standard deviation σ_{error}, in part because this is a good estimation for many measurement processes and in part because of the mathematical convenience associated with Gaussian distributions. The probability density function (PDF) of sensing z_t is hence given by a normal distribution $\mathcal{N}$ (see also Section 3.20):

$$p_{\text{error}}(z_t|s_t, m) = \eta\, \mathcal{N}(z_t, z_t', \sigma_{\text{error}})$$

The normalizing constant η is calculated by integrating over the sensing range and $p_{\text{error}} = 0$ outside the sensing range. p_{error} is the "classic" error of measurements used in physics assuming an external experimenter that can discard erroneous measurements and outliers caused by systematic errors discussed below. Since the robot has to cope without this help, unavoidable systematic errors must be included in the sensor model.

Measurement failures, $p_{\mathbf{max}}(z_t|s_t, m)$ If the sensor fails to detect an object although z_t' lies within the sensing range, the sensor wrongly measures a

range of $z_{\max}$. For range finders, this happens if a specular reflection on the object surface causes the beam to bounce off the object away from the sensor. The lack of a return signal produces a measurement $z_{\max}$. These events can either be modeled as a delta peak in the PDF at $z_{\max}$ or such measurements can simply be discarded as they are easily identified.

Unexpected objects, $p_{\mathbf{short}}(z_t|s_t, m)$ Dynamic objects (people, other robots, etc.) that are not contained in the map and that move freely in the robot's environment may occlude other objects and features and result in a range reading shorter than $z_t < z'_t$. I will show later how occlusion can be modeled explicitly when using a camera as the primary sensor and how this can be used to ascertain negative information. For range scanners, however, occlusions are hard to detect in the unprocessed sensor data. They can be tackled by explicitly including objects in the state vector, estimating their location and modeling occlusions. Occlusions also can be integrated directly as sensor noise. The probability of an object in the way of the beam is given by an exponential distribution that is cut off at z'_t.

Unexpected measurements, $p_{\mathbf{rand}}(z_t|s_t, m)$ Under this type of noise, hard to model effects and unexplainable measurements are subsumed. This noise may be due to higher order physical effects such as sonar being reflected off multiple walls, cross-talk between sensors of one robot or between multiple robots, etc. Such noise is modeled as being uniform over the sensing range and zero outside.

These four types of noise are used to model the noise inherent in range scans. Note that for other sensors, other types of noise may be observed and need to be modeled accordingly. All sensor have in common the measurement error and the unexpected measurement error.

The probability of a sensor reading z_t is given by the weighted average of all types of noise:

$$p(z_t|x_t, m) = \sum_i \alpha_i \; p_i(z_t|x_t, m) \tag{6.4}$$

where i runs over the types of noise and the weights α_i sum up to 1. The model parameters of the above sensor model are: the weights α_i, the standard deviation σ_{error} of the measurement error, and γ_{short}, a parameter describing the probability of measuring ranges shorter than the true range z'_t. Finding these model parameters is done by fitting the model to typical measurement data. While this may sound straightforward, it is not always easy to identify a *typical* data set. The probability of objects being occluded, for example,

depends on the position of the robot in its environment, the number of dynamic objects currently present in the environment and their current dynamics, and also on the agent's current task (when chasing an object, the robot may have to deal with more unexpected readings than in a navigation task). Such an analysis also requires the true range z_t' to be known, requiring accurate external monitoring of the robot, which is not always available. For these reasons, model parameters are often guessed by the experimenter, who has to know and decide which data can be regarded as typical and who has to trade off particle filter robustness against accuracy. In general, Monte Carlo methods have been found to be fairly robust with respect to specific sensor modeling, producing good results even if the sensor is modeled crudely. Despite of what has been said, parameters can be identified from the data by maximum likelihood estimation, as long as data is carefully selected.

A range scanner commonly produces an array of K values at time t, $z_t = \{z_t^1, ..., z_t^K\}$. The probability of each sensor reading is used to calculate the probability to the current measurement:

$$p(z_t|s_t, m) = \prod_{k=1}^{K} p(z_t^k|x_t, m) \tag{6.5}$$

This assumes the noise of individual measurements to be independent, similar to how – under the Markov assumption – noise is assumed to be independent over time. In feature-based sensor models like the one used in our experiments, the above product runs over detected features.

In feature-based measurement models, a level of abstraction from the raw sensor data is achieved by *feature extraction*. In the case of a range finder, such a feature extractor could find corners of walls in the array of range measurements. For camera images, there are countless types of feature extractors using the two-dimensional array of intensity values (i.e., the image) and processing it to find edges, corners, pattern, shapes, etc. Features often correspond to distinct objects in the robot's environment, called *landmarks*. The extraction of a comparatively small number of features from the high dimensional space of sensor measurements yields a significant reduction of computational complexity of the process of inference.

The vision system employed by the GermanTeam extracts the colored beacons and goals from camera images. Knowing the current gaze direction of the robot relative to the robot's body, the range r and bearing ω of landmarks relative to the robot can be calculated from the feature's position in the camera image. The actual implementation of the localization module takes into account only the bearing to a landmark (see Fig. 6.1). Range measurements were found to be very unreliable because a partially occluded landmark appears to be smaller in the camera image and can thus lead to a faulty range

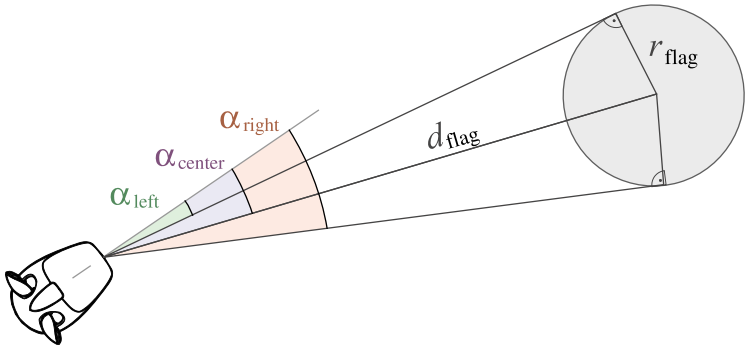


Figure 6.1: Top down view of robot looking at a flag. For localization, the angles to the left and right corner of a flag are used, ω_{left} and ω_{right}, but not the angle to the landmark's center (which is equivalent to the bearing). ω_{left} and ω_{right} are treated as independent measurements.

measurement. Therefore, range data is disregarded altogether.

In addition to the bearing to the landmark, its type τ_t is also stored (i.e., one of the four beacons or one of the two goals; this is also called the feature's *signature*). Similar to 6.5, the probability of detecting the set of features $f(z_t) = \{f_t^1, f_t^2, ...\}$ is given by:

$$p(f(z_t)|s_t, m) = \prod_i p(\omega_t^i, \tau_t^i|s_t, m) \tag{6.6}$$

We assume that landmarks are unique and can be uniquely identified, that is to say, the data association problem is solved. The sensor model for detecting a specific landmark using a feature based map is defined by:

$$\begin{aligned} p(\omega_t|s_t, m) &\equiv p(\omega_t, \omega_t') && (6.7)\\ &= \eta\, \mathcal{N}(\omega_t, \omega_t', \sigma) && (6.8)\\ &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(\omega_t - \omega_t')^2\right) && (6.9) \end{aligned}$$

Here, the true bearing ω_t' to the landmark at (x_m, y_m) is computed using the current robot pose (x_t, y_t, θ_t):

$$\omega_t' = \arctan\left(\frac{x_m - x_t}{y_m - y_t}\right) - \theta \tag{6.10}$$

The error described in Equation 6.9 subsumes the following errors, which are all assumed to be Gaussian:

Limited camera resolution The camera resolution is limited and images are color segmented. The width of blobs therefore has integer value and a typical error of ± 1 pixel. The bearing to objects is only slightly affected by this, while for range measurements, this effect increases with distance to the object. It amounts to a large error for far away objects, which appear small in the camera image and thus have an unfavorable signal-to-noise ratio. The bearing error is of the order of the ratio of camera opening angle and camera resolution, $\Delta\omega_{\text{cam}} \approx 0.5°$.

Aberrations Other sources of error are chromatic and spherical aberration of the camera. The former may lead to classification errors while the latter means that the actual projection differs from the pin hole model. Both types of error can be addressed by camera specific calibration [Röfer et al., 2005].

Motion blur Quick movements of the robot and of its head may cause the image to become blurry, making it hard to detect feature borders. In particular, objects may appear to be compressed in the direction of camera movement. This error is estimated to be $\Delta\omega_{\text{blur}} \approx 10°$.

Joint slackness The joint slackness of the head joints of the Aibo introduce another source of error. Since knowledge about the current value of the joints is necessary to perform the transformation from the camera's to the robot's frame of reference, the uncertainty about exact joint positions translates into a bearing error. The uncertainty increases when the head is moving fast. $\Delta\omega_{\text{slack}} \approx 15°$.

Joint slackness and motion blur by far outweigh the error introduced by limited camera resolution. Nisticò and Röfer [2006] show how this error can be modeled taking into account the dynamics of the robot and its head and [del Solar and Vallejos, 2004] describes how camera motion can be estimated by image alignment to allow object tracking. In our studies, these effects were not explicitly modeled and we used a static model with $\sigma_{\text{total}} = \sqrt{\sum \Delta\omega_i^2} \approx 20°$. This parameter value has been tested in many experiments and RoboCup games, ensuring good performance and sufficient robustness.

The model does not explicitly include dynamic objects and possible occlusions caused by them. When occlusions occur, the feature extractor will simply fail to detect the landmark and the belief will not be updated. This differs fundamentally from integrating range finder data, where measurements lack signature and it is not possible to easily disregard measurements brought about by occlusions.

Note also that the sensor model contains an error that is caused by robot motion. If the goal was to track the exact position of the camera in space, this

error would, in fact, be included in the motion model. Since we are primarily interested in the position of the robot, it is safe to include it in the sensor model.

Reducing the raw sensor data to features comes at the price of not using the available information to the fullest. While this might be a computational necessity, one needs to keep in mind that available information in the camera images is not used, sometimes leading to ambiguities and a decrease in performance.

6.1.2 Related Work

Although robot localization and in particular Monte Carlo Localization is of continuing interest to many researchers, the concept of negative information has not been used for localization. Some recent work does use negative information in object tracking, an illustrative introduction of which can be found in a tech report [Särkkä et al., 2004]. The event of not detecting an object is treated as evidence that can be used to update the object's probability density function in [Agate et al., 2004; Koch, 2004]. In the RoboCup domain, not seeing the ball on the field is used in [Kwok and Fox, 2005] to update the probability distribution in that region; occlusions are dealt with using a model of the other robots on the field, which needs to be maintained and is less reliable then using visual cues in the image.

Negative information is also mentioned in the context of *simultaneous localization and mapping* (SLAM). It is used to adjust the confidence in feature candidates and remove features that the robot could not detect at later times [Montemerlo and Thrun, 2003].

6.2 The Notion of Negative Information

The classic example of negative information is described in the Sherlock Holmes case "Silver Blaze." In this case, a house has been broken into. Under such circumstances, one would expect the watch-dog to bark. The curious incident of the non-barking of the dog at nighttime provides Holmes with the information that the dog must know the burglar, allowing Holmes to solve the case. Applied to mobile robot localization, this means that conclusions can be drawn from expected but actually missing sensor measurements [Koch, 2004].

This research focuses on localization based on features (landmarks). Whenever a robot senses a feature, the localization estimate is updated using the sensor model. Sensor updates only occur when features are detected in existing approaches. If no feature is detected, the state estimation is updated using (only) the motion model of the robot.

Example Consider a robot driving down a corridor as shown in Fig. 6.2. The robot has a sensor to detect doors when it is standing in front of one. Let us assume further that the robot is moving to the right but is oblivious of its starting position. As it starts to move to the right, it passes and senses a door. Given this information, it could be standing in front of either of the doors (states s_{left} and s_{right}). As it moves on, it does not pass another door for some time. At time $t = t_3$, if s_{left} had been the true position, the robot would have had passed the other door by now. Using the negative information of not perceiving a door, the belief based on s_{left} can be ruled out. As Thrun, Bugard, and Fox put it quite graphically, "not seeing the Eiffel Tower in Paris implies that it is unlikely that we are right next to it" [Thrun et al., 2005].

Definition *Negative information* marks the ascertained absence of an expected sensor reading. The term is chosen in accordance with the terms *(false) positive* and *(false) negative* used in statistics. A *false positive* denotes a "ghost measurement," in other words, measuring something that is not really there. A *false negative* describes the incorrect absence of an observation when a sensor fails to detect something that is actually there.

Negative information constitutes a smaller information gain per update than sensing a landmark since, in general, there are fewer locations from which a landmark is visible (i.e., high information gain when a landmark is detected) than positions from where it is not (i.e., low information gain when no landmark is detected). A landmark is, by definition, something that stands out in an environment and thus yields an information gain. However, as the robot moves about in its environment, negative information can be integrated over time and can yield significantly improved localization performance. This is especially true for cases where the robot cannot focus on landmarks because of its current task. In the above definition, it is important to note that the absence of a sensor reading needs to be ascertained. The non-observation on a real robot has three main reasons:

1. The object is outside the sensor's sensing range.
2. The object is occluded.
3. Imperfect sensor data or faulty image processing may keep the robot from detecting the object.

Differentiating the first two cases requires careful sensor modeling. This problem is addressed by considering the *field of view* of the robot and by using obstacle detection to estimate occlusions. The third case can be tackled using a probabilistic sensor model, that is, by modeling the probability of the sensor detecting an object that is within its sensing range and not occluded.

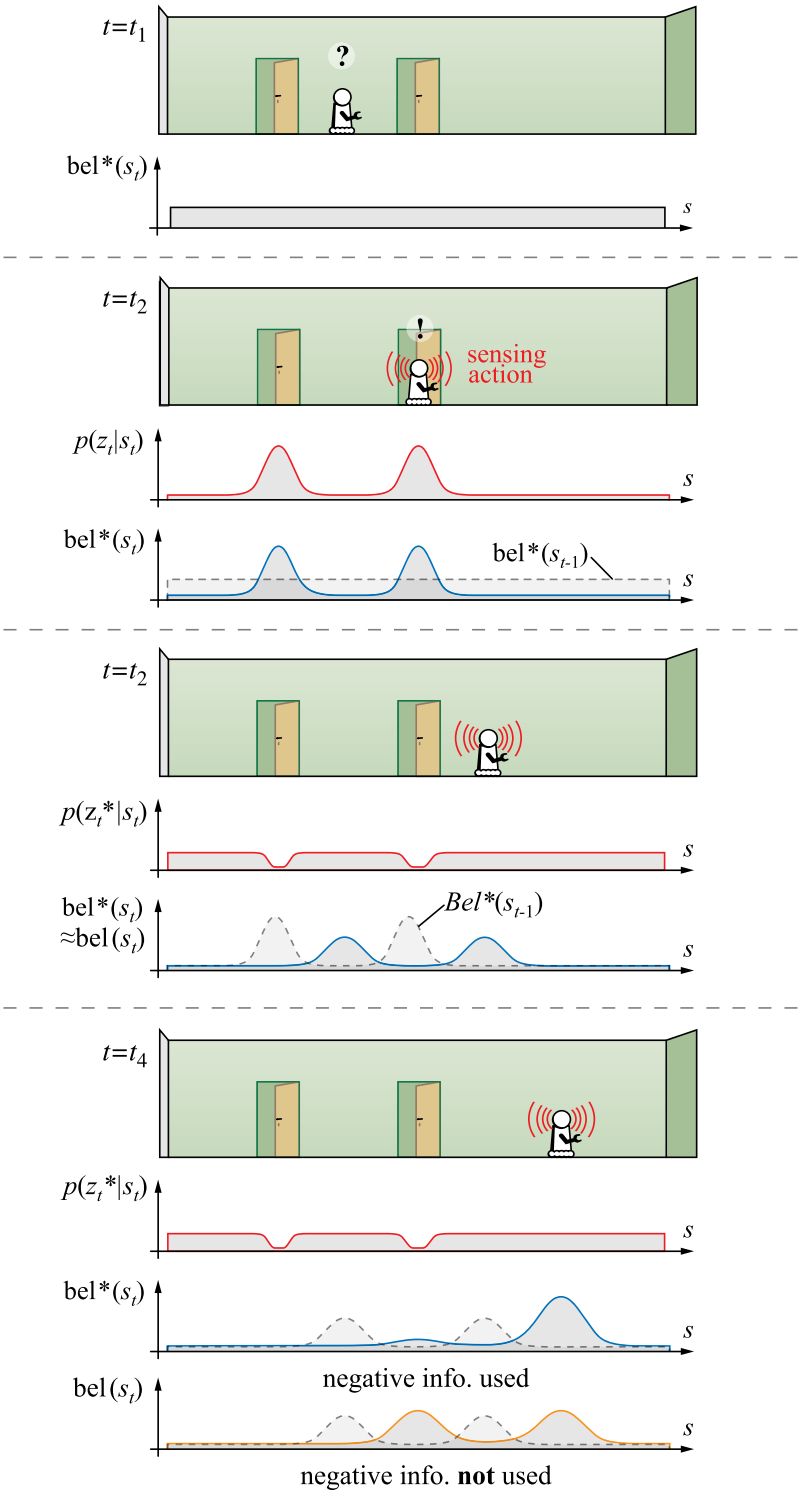


Figure 6.2: Thought Experiment I: Robot localizing in a hallway.

6.2.1 Robot Localizing in Hallway

Let us first return to the example of the robot localizing in a hallway and examine it in more detail (Fig. 6.2). The robot has a sensor to detect doors and moves along the hallway. It is limited to movement in one dimension. Let us further assume that it is moving to the right and starts off at an unknown position in the hallway. The robot is also equipped with a map of the hallway and the sensor is modeled using the sensor model $p(Z = z_t|s_t, m)$.

$t = t_0$ In the beginning, the robot does not know its position in the hallway, as is described by the uniform belief distribution $\text{bel}^\star(s_t)$. At this time, no sensing of the world takes place. The robot starts moving to the right.

$t = t_1$ The robot has moved down the hallway and now senses a door $p(z_t|s_t)$, which results in the shown belief $\text{bel}^\star(s_t)$. It has two peaks since the robot could be standing in front of either door. The previous distribution is shown by the dashed line. The robot continues to travel to the right.

$t = t_3$ At this position, there is no door for the "door sensor" to detect. The sensor update is described by $p(z_t^\star|s_t)$. The negative information is of little use at this position: it does not help the robot differentiate between the peaks. Note that the probability density becomes more smeared out due to the uncertainty brought about by the robot's motion.

$t = t_4$ The robot moves on and the door sensor still does not detect a door. $\text{bel}^\star(s_t)$ shows the belief if negative information is taken into account, whereas $\text{bel}(s_t)$ shows the belief without using negative information to better illustrate the case. As can be seen from the diagram, using negative information allows the robot to rule out the left peak. Note that the left peak is not completely eliminated since there is a non-zero likelihood of the robot not sensing a door even if it is standing in front of it.

6.2.2 Robot Figuring out its Orientation

The following example illustrates two aspects of negative information. One aspect is that for a given position x_t, the probability of sensing a landmark and the probability of not sensing a landmark add up to one.

$$p(Z_t = \text{'landmark'}|x_t) + p(Z_t = \text{'no landmark'}|x_t) = 1 \tag{6.11}$$

The probability $p(Z = \text{'no landmark'}|x_t)$ is the complementary probability to $p(Z = \text{'landmark'}|x_t)$. The other aspect is that the immediate information gain of negative information is not very large. As negative information is

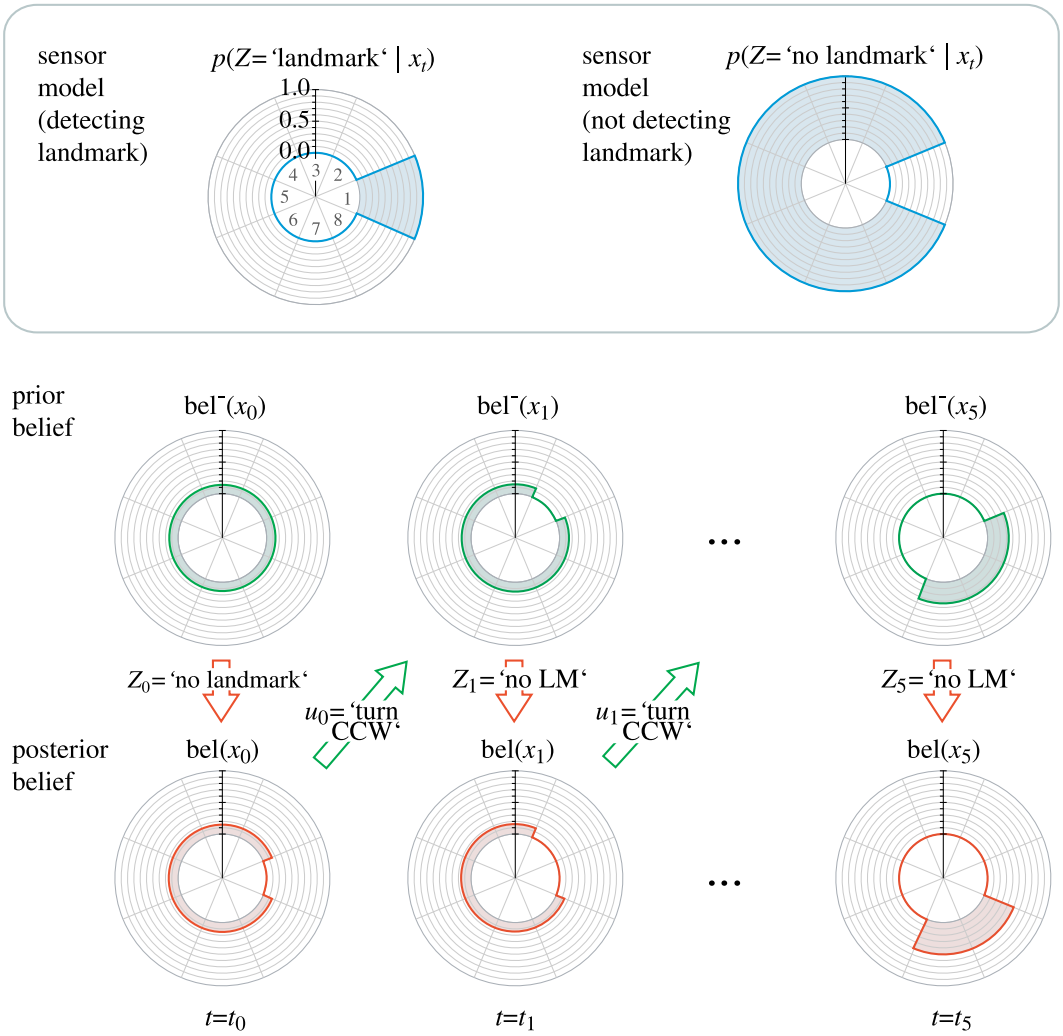


Figure 6.3: Thought Experiment II: In a discrete world, an agent tries to figure out its current orientation from the set of 8 possible orientations by turning counter-clock-wise (CCW). There is only one landmark in the environment, the sensor models for sensing and for not sensing the landmark are illustrated at the top. As the robot turns but does not detect the landmark, its belief is updated using negative information.

integrated over time, however, a significant improvement of localization can be achieved.

Imagine an agent that can only turn by discrete steps of $\pi/4$ Fig. 6.3. It possesses a sensor that can detect a landmark in the sector in front of it. There is only one landmark in the robot's environment; actions and sensing are assumed to be deterministic. In Fig. 6.3 (top), the sensor model for detecting

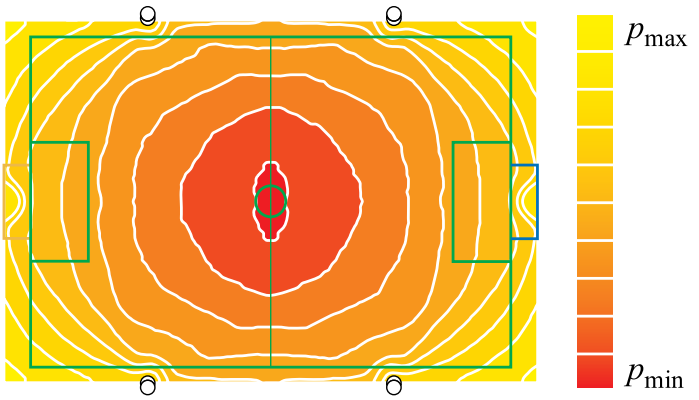


Figure 6.4: Probability of *not* sensing a landmark for a robot on a RoboCup soccer field. For a robot located around the center of the field, it is hard to miss landmarks.

and for not detecting a landmark is shown. The probability of the robot to be facing in the direction $X_t = x_t^\star$ where it cannot see the landmark, i.e., $x_t^\star \in \{2, 3, 4, 5, 6, 7, 8\}$, is calculated using the sensor model ($z_t^\star$ is the non-detection observation):

$$p(x_t^\star|z_t^\star) = \frac{p(z_t^\star|x_t^\star)\ p(x_t^\star)}{p(z_t^\star)} \tag{6.12}$$

$$= \frac{1 \cdot 1/8}{7/8} = 1/7 \tag{6.13}$$

The agent starts off not knowing its orientation at $t = t_0$, i.e., uniform belief. The first sensor measurement yields no landmark detection. The robot starts turning ("scanning for the landmark"), which is integrated into its belief as a motion update. In each of the following steps, the landmark is not detected. At $t = t_5$, the robot has almost completed one full rotation. The probability distribution has converged to the two sectors shown in Fig. 6.3. Such convergence only takes place when negative information is integrated into the belief. Discarding negative information, as it is done in previous localization approaches, results in a uniform belief that is unchanged from the beginning of the run.

If the robot turns one more step at $t = t_6$ and then does not detect the landmark, it is certain to face in direction of sector 8, without ever having seen the landmark.

1: **Bayes-Filter**$(\text{bel}(s_{t-1}), u_t, z_t)$
2: $\quad \text{bel}^-(s_t) \longleftarrow \int p(s_t|s_{t-1}, u_{t-1})\text{bel}(s_{t-1})ds_{t-1}$
3: $\quad$ **if** (landmark l detected) **then**
3: $\quad\quad \text{bel}(s_t) \longleftarrow \eta p(z_t|s_t)\text{bel}^-(s_t)$
4: $\quad$ **else**
5: $\quad\quad \text{bel}(s_t) \longleftarrow \eta p(z^\star_{t,l}|s_t, r_t, o_t)\text{bel}^-(s_t)$
6: $\quad$ **endif**
7: $\quad$ **return** $bel(s_t))$

Figure 6.5: Pseudo code of Bayes Filter incorporating negative evidence.

6.3 Exploiting Negative Information in Robot Localization

6.3.1 Generic Sensor Model

Negative information describes the absence of a sensor reading in a situation where a sensor reading is expected given the current position estimate. To integrate negative information, imagine a binary sensor being added that fires whenever the primary sensor *does not* detect a particular landmark l. Its probability of firing is given by:

$$p(z^\star_{l,t}|s_t, m) \tag{6.14}$$

This sensor model can be used to update the robot's belief whenever it fails to detect a landmark, i.e., when negative evidence is acquired. Fig. 6.4 shows the probability $p(z^\star_t|x_t, y_t, m_{\text{field}})$ of not sensing a landmark on a RoboCup field described by map m_{field} at position (x_t, y_t) summed over all possible robot orientations. This figure also shows that it is most likely for the robot to sense a landmark when it is standing in the middle of the field. The likelihood of not sensing a landmark is highest for positions at the edge of the field as the robot may be facing outwards.

This rather coarse way of incorporating negative information can be refined by taking into account the sensing range r_t of the robot's sensors and possible occlusions o_t of landmarks. The sensing range is the physical volume monitored by the sensor. In case of a stationary sensor, $r_t = r_0$ is constant, for a mobile robot with a pan-tilt camera it is a function of the current robot pose and gaze direction. By o_t we denote a means of detecting whether or not occlusions have occurred. In practice, this can be calculated from a map of the environment, directly sensed by a sensor such as a laser range finder, or derived from a model of moving objects in the environment.

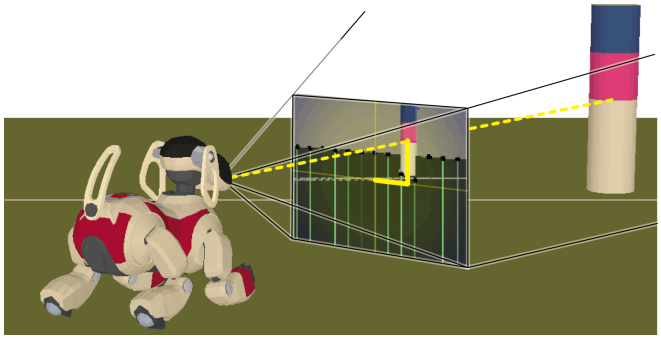


Figure 6.6: Illustration of the viewing frustum of a Sony Aibo facing a landmark on a RoboCup field. Also indicated: the projection of the landmark onto the camera plane.

Combining the two yields the probability of not sensing an expected landmark l:

$$p(z^{\star}_{t,l}|s_t, r_t, o_t, m) \tag{6.15}$$

Whenever a landmark is not detected, it can be used in the sensor update step of the Bayes Filter described in Fig. 6.5.

6.3.2 Sensor Model for the Camera of the Sony Aibo

Sensing Range

The Aibo ERS-7 has a camera built into its head. This camera has a horizontal opening angle of $\gamma = 55°$ and the robot's head has 3 degrees of freedom (neck tilt, head pan, head tilt). Gaze direction is abbreviated by $\varphi = (\varphi_{\text{tilt1}}, \varphi_{\text{pan}}, \varphi_{\text{tilt2}})$. The sensing range is illustrated in Fig. 6.6 and is determined by these three angles, the current *robot pose* (position and orientation), and the the camera's viewing frustum/field of view. An object can only be visible to the robot if it lies within camera's viewing frustum. This can be determined using the object's bounding box.

Occlusions

To account for occlusions, the obstacle percept used to build up the obstacle model described in Chapter 4 is employed. The camera image is scanned in vertical scan lines and unoccupied space in the plane of the field is detected since it can only be of green or white color (field lines). Scanning for these colors provides information about where there is free space and where there are obstacles. This information is stored in the *obstacle percept*, which provides

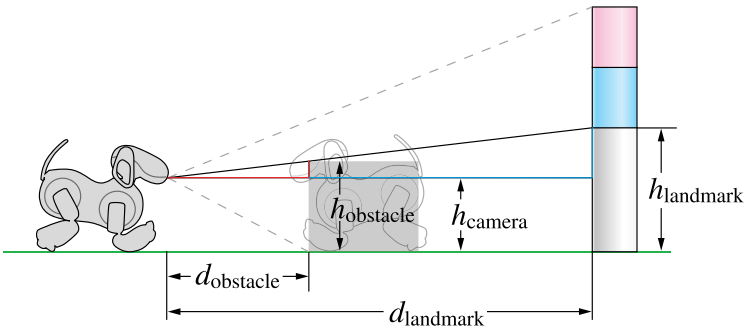


Figure 6.7: Occlusions caused by other robots. The obstructing robot is abstracted by the gray box. If it was to move further to the left, it would partially occlude the landmark.

the basis for determining if the visibility of the landmark is impaired, i.e., if it is occluded by other robots or some other obstacle. More specifically, if the expected landmark lies in an area where the robot has detected free space, the likelihood of the corresponding pose estimate is decreased. If it lies outside of the detected free space, no inference can be made, the landmark may be occluded or outside the sensing range.

In the RoboCup environment, occlusions by the static environment do not occur unless the robot leaves the field, so they can safely be neglected. The only cause of occlusions are other robots on the field. An important performance factor of the algorithm is the ability of the vision system to differentiate between obstacles that can actually cause occlusions and those which cannot.

On the other hand, a robot in front of a landmark does not necessarily occlude it, as is shown in Fig. 6.7. Based on the intercept theorems, occlusion occurs only if $\frac{(h_\text{o}-h_\text{c})*d_\text{l}}{d_\text{o}} - h_\text{c} > h_\text{l}$. One important result of this is that obstacles farther away than 1m cannot occlude landmarks on the field.

Considering the viewing frustum and possible occlusions, the sensor model for not perceiving an expected landmark (Equation 6.15) is written as:

$$p(z^\star_{t,l}|s_t, o_t, m) \tag{6.16}$$

where o_t describes the current *obstacle percept* and $s_t = (x_t, y_t, \theta_t, \varphi_t)$ the robot state consisting of the *robot pose* (position x_t, y_t, and orientation θ_t) and the current gaze direction φ_t.

The probability is modeled as follows: if the expected landmark is not detected and lies outside the viewing frustum, nothing can be inferred and the probability is set to 1. To make the sensor model more robust with respect to joint slackness and similar effects, the opening angle used in the visibility check is smaller than the actual opening angle, $\gamma' = \gamma - \Delta\gamma$. If the expected

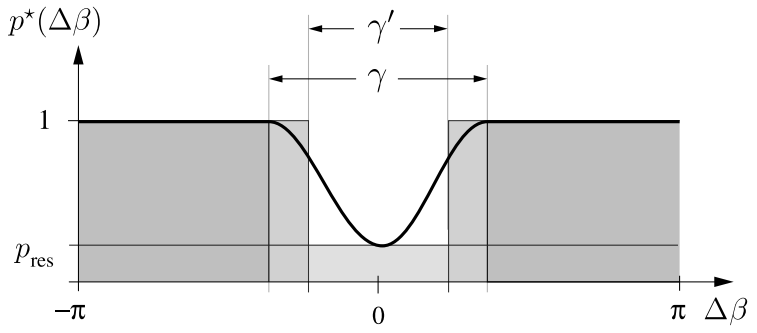


Figure 6.8: Sensor model used for negative information. The area outside the viewing cone (dark shade of grey, opening angle γ) have probability 1, i.e., if a landmark is expected outside the viewing cone and is not detected, nothing can be inferred. γ' can be understood as a safety buffer accounting for joint imperfections and uncertainties due to the dynamics of the robot's head. Residual probability p_{res} represents the non-zero probability of the sensor missing an unobstructed object within the viewing cone. The solid line shows the smooth approximation of the sensor model.

landmark is occluded as determined by the obstacle percept, still nothing can be inferred and the probability remains 1. If, however, the expected landmark is within the viewing cone and the landmark should be in unobstructed, plain sight, the probability of this hypothesis is adjusted. One could simply set the probability of that particle to zero. However, particle filters work better for smooth sensor models; therefore, the sensor model shown in Fig. 6.8 is used. The angle $\Delta\beta$ between gaze direction $\vec{g}(\varphi)$ and the vector to the landmark $\vec{l}$ in camera coordinates is calculated using the dot product. The sensor model thus becomes:

$$p(z^\star_{t,l}|s_t, o_t, m) = \begin{cases} 1 & \textit{if } \text{landmark is outside FOV} \\ & \textit{or } \text{landmark is occluded} \\ 1 \geq p^\star(\Delta\beta) \geq p_{\text{res}} & \textit{if } \text{landmark expected in plain sight} \end{cases} \tag{6.17}$$

with

$$p^\star(\Delta\beta) = 1 - (1 - p_{\text{res}}) \exp\left(-\frac{1}{2\sigma^2}(\Delta\beta_t)^2\right) \tag{6.18}$$

$$\Delta\beta = \arccos\left(\frac{\vec{g}\cdot\vec{l}}{|g|\ |l|}\right) \tag{6.19}$$

$$\sigma = \gamma'/2 \tag{6.20}$$

6.4 Experimental Results

As before, the RoboCup Sony 4-Legged League serves as a test bed for our research. Recall that the colored landmarks (4 uniquely color coded beacons plus a blue and a yellow goal) and the field lines (similar to the real soccer field lines) serve the robots for localization. In our experiments, unless otherwise stated, only landmarks were used for localization to emphasize the effect of negative information.

Monte Carlo Localization

The approach is based on MCL described in Section 3.5. Sensor update is extended to account for field of view and occlusion as described. This also requires sensor updating to be triggered by new camera images regardless of whether or not there is a percept. Before re-sampling, the weight of an individual particle is calculated as follows: Of all landmarks L, the subset of landmarks $L^\dagger$ is detected, the subset $L^\star$ is expected but not detected, and lastly the subset $L^\diamond$ is not detected but is also not expected: [1] $L = L^\dagger \cup L^\star \cup L^\diamond$ and $L^\star \cap L^\dagger = \emptyset$. The weight w_i (or probability) of a particle i is calculated by multiplying all the likelihoods of all gathered evidences:

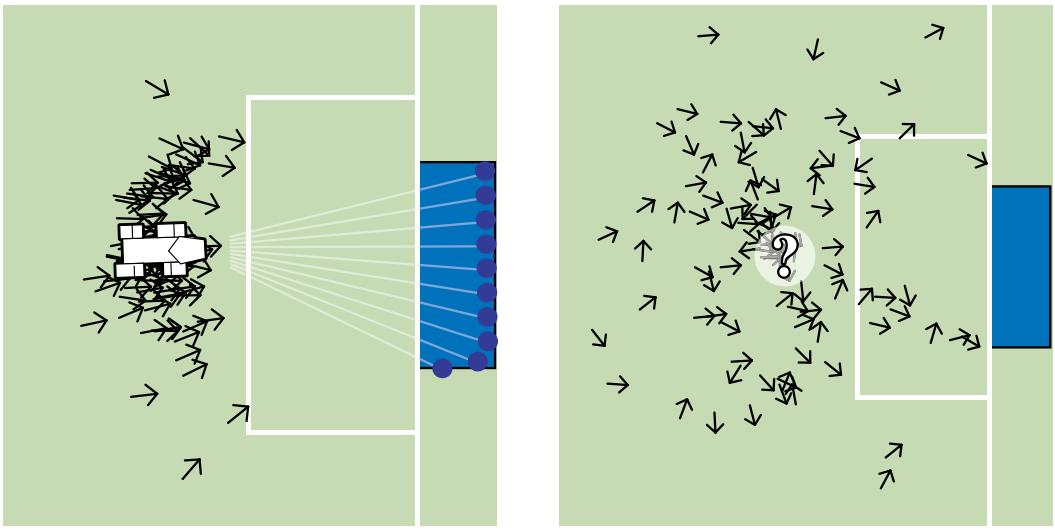
$$w_i = \underbrace{\prod_{l \in L^\dagger} p_l(\omega_t, \omega'_t)}_{\text{detected}} \cdot \underbrace{\prod_{l \in L^\star} p^\star_l(\Delta\beta_t)}_{\text{exp'd but not detected}} \tag{6.21}$$

The function p_l is the sensor model introduced in Equation 6.9 and describes the likelihood of sensing the landmark l at angle ω_t for a particle p_i that expects this landmark to be at ω'_t. Function $p^\star_l$ models the probability of not sensing the expected landmark $l \in L^\star$ given the current sensing range as determined by gaze direction φ, the robot pose associated with particle i, and the obstacle percept z_{obs} (Eqn. 6.17).

In sensor updates, the rate of change of the weight of a particle that is brought about by the sensor model is artificially limited. This is necessary because of the small number of particles used and the unreliable sensor data. If this is not done, a single mis-processed image can cause the distribution to collapse or to become unsteady [Röfer and Jüngel, 2003]. This is the reason why not sensing an expected landmark as shown in Fig. 6.10 does not immediately eliminate all particles that receive negative information but does so over time (consecutive frames without landmarks).

Two quantities can be used for localization when a landmark is seen: its size in the camera image can be used to estimate the distance to the landmark

[1] False positives are not considered.



1) 2)

Figure 6.9: 1) Robot is standing in front of the goal facing it, being well localized. The lines facing away from the robot's head indicate that the robot is looking at the goal and the blue circles show detected parts of the goal. Each arrow represents a Monte Carlo particle. The maximum of the probability distribution corresponds to the actual robot position. If the robot is being kidnapped and no noise is added to the particles, the distribution will remain as it is. If noise is added over time (to model unforeseen external events such as kidnapping), the probability distribution will spread out (2) and in the limit $t \rightarrow \infty$ will become uniform.

d_l and the relative angle to the landmark (bearing, α_l) can be calculated from its position within the image. In practice, only the bearing is used because the distance measurement is error prone. Using just the bearing, only the orientation of the robot can be inferred from a single landmark. Note that this differs from triangulation where distances are used.

Differences To Standard-GT-MCL The implementation of the MCL module differs from the one used by the GermanTeam in a number of aspects. The changes aim at making the particle filter behave more like a standard "text book" version of MCL (see Section 3.6):

- The particle count is increased to give smoother results and to reduce artifacts due to the small number of particles.
- Noise is added even when no observation is made.

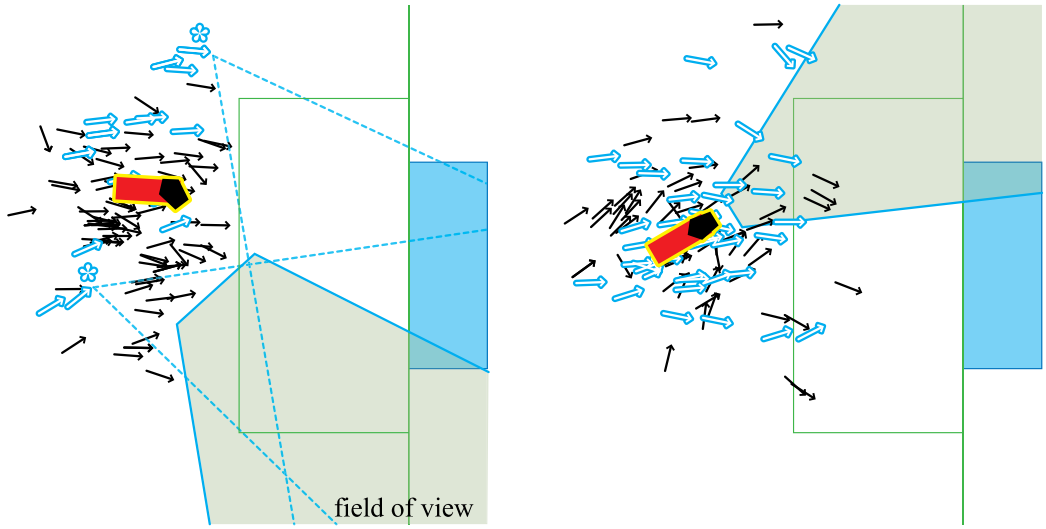


Figure 6.10: Incorporating negative information. Blue-white (outlined) arrows denote particles that receive negative information. In 1), the effect of using negative information is shown for a robot that is well localized and frequently sees landmarks. The shaded area represents the sensing range of the camera for the actual robot (the robot does not currently perceive the goal). The viewing frustums for two hypothesis (marked each with a star) are also shown. If either of them reflected the actual robot's pose, the robot would be able to see the goal. 2) The distribution shortly after the robot has been displaced (kidnapped): particles representing poses facing the goal are less likely and will eventually be eliminated from the distribution.

- *Templates*[2] are not used.
- Field lines are not used for localization unless otherwise stated.

Performance Measures

The experiments were conducted in simulation using log files of real camera images and of simulated images to ensure reproducible results and identical sensor input when benchmarking the approach. This also allows for the use of a greater particle count, which results in a smoother representation of the probability distribution (this is mainly for illustration purposes). Let us briefly recapitulate the means of characterizing particle distributions described in more detail in Sections 3.7:

[2] a.k.a. *sensor resetting*, see Section 3.6.2

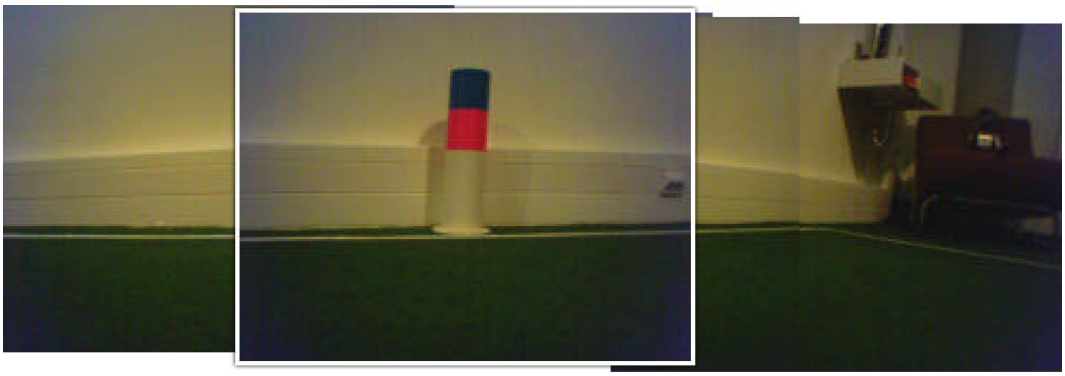


Figure 6.11: *Experiment E1:* A panorama view generated from actual camera images, single camera image highlighted. The robot can only see *one* landmark.

Entropy The expected entropy H is used as an information theoretical quality measure of the position estimate $\text{bel}(s_t)$ [Fox et al., 1998a]:

$$H_p(s_t) = -\sum_{s_t} \text{bel}(s_t) \log(\text{bel}(s_t)) \tag{6.22}$$

The sum runs over all possible states. The entropy of the particle distribution becomes zero if the robot is perfectly localized in one position, maximum values of H mean that $\text{bel}(s_t)$ is uniformly distributed.

Localization Error In the experiments, the distance error of the localization with respect to the robot's real position, which is easily accessible in simulation, is also used. The error Δr is defined as the average distance of particles to the actual robot position $\vec{x}_{\text{true}}$:

$$\Delta r = \frac{1}{N} \sum_{i=1}^{N} |\vec{x}_i - \vec{x}_{\text{true}}| \tag{6.23}$$

where $\vec{x}_i$ is the position of particle i and N is the number of particles.

Similarly, the expected entropy and accuracy of the robot's *orientation* are calculated.

Preliminary Experiment

For illustration purposes, a preliminary experiment was conducted in simulation. In this experiment, the robot starts out being well localized and is then displaced to a position where it is not able to get any new sensor information (Fig. 6.9 and 6.10). It is similar to the kidnapped robot problem, but here the moment right after the robot is displaced is emphasized. The effect

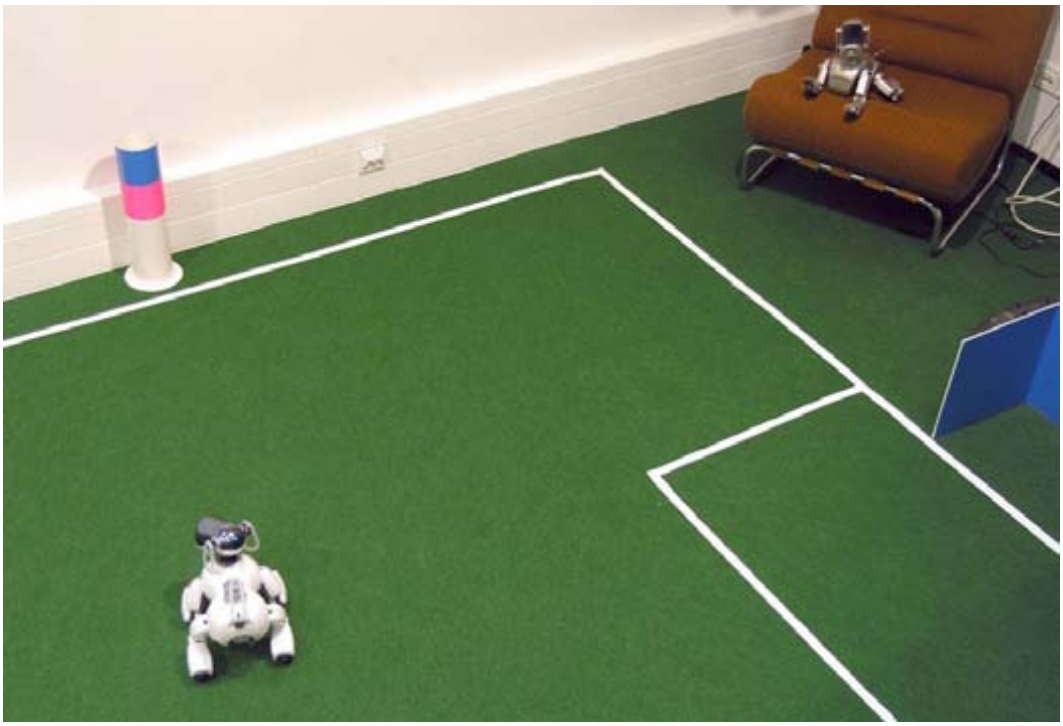


Figure 6.12: *Experimental setup E1:* Robot is standing at the position shown in the photo. It performs a scanning motion with its camera.

of the displacement on the Monte Carlo particle distribution is the following: particles which represent the previous belief become less likely when negative information is taken into account. The distribution diverges towards particles which were less likely prior to the displacement. Particles representing the previous belief are eventually eliminated from the distribution because they are inconsistent with the current (negative) sensor data. Particles which differ from the previous belief just enough to be compatible with the current sensor data are favored; particles remain close to where the robot was last able to localize.

6.4.1 E1 Localization Experiment

In the following experiments, the localization module starts with a uniform (random) particle distribution (uniform *a priori* belief). As the robot receives sensor measurements, the progression of the distribution is monitored.

The robot is positioned in front of a single landmark, standing still but performing a scanning motion with its head (Fig. 6.12). This scan covers $90° + 55° = 145°$ in front of the robot. Within this area, there is but one landmark. A panorama image composed of actual robot camera images is shown in Fig. 6.11.

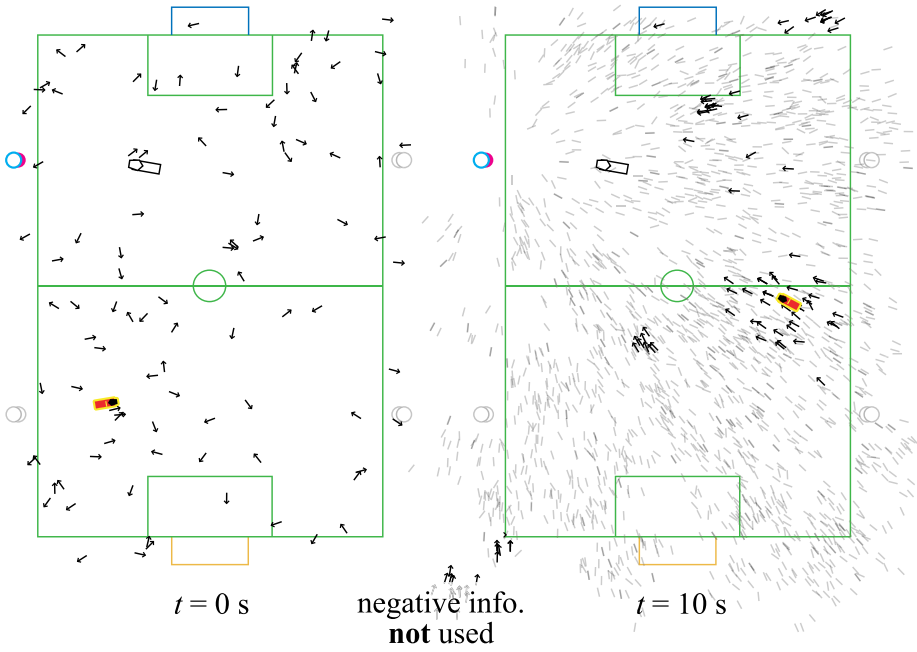


Figure 6.13: *Experiment E1:* Particle distribution not using negative information, initial uniform distribution (left) and distribution after 10s (right). Solid arrows indicate Monte Carlo particles (100). The experiment was repeated using 2000 particles (shaded lines). The actual robot position is indicated by the symbol in the top left half of the field, the estimated robot pose by the solid symbol. Not using negative information and only using the bearing to the landmark, the robot is unable to localize. Some clusters of particles form, but they do not converge. As one would expect, the position distribution is almost uniform but the relative angle to the seen landmark is distinctly visible in the distribution.

The goal of this experiment is to show that even with very few sensor readings, localization is possible when negative information is also taken into account.

Particle Distribution

In the following paragraphs, the basic localization not using negative information and localization incorporating negative information are compared. First, the particle distribution is analyzed qualitatively and then it is shown how the entropy of the distribution decreases when negative information is considered.

The basic experiment was conducted using 100 particles for Monte Carlo localization. It was repeated on a log file containing camera images, robot joint angles, and odometry data using an increased particle count of 2000 to

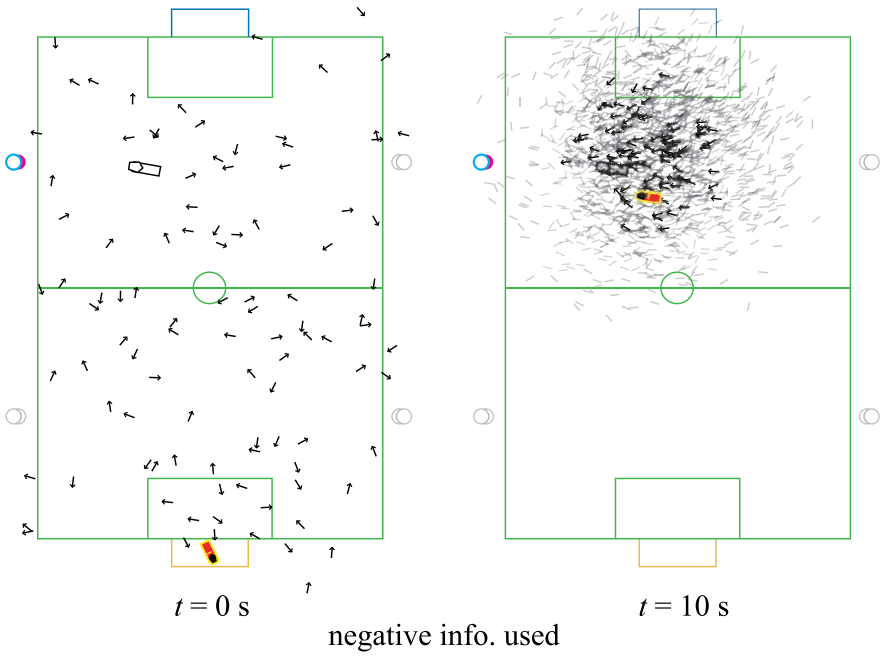


Figure 6.14: *Experiment E1:* Particle distribution when negative information is incorporated (cf. Fig. 6.13). When incorporating negative information, the robot is able to localize in a position where it otherwise – using only landmark detections – is unable to localize.

produce smoother representations of the probability distributions.

Not using negative information. Without using negative information, the robot is unable to localize (Fig. 6.13). Only the orientation of the particles is adjusted according to the sensor readings. The apparent clustering in the small sample set in Fig. 6.13 is not stable and, even after considerable time, the particles do not converge. The distribution for the larger sample set is also uniform (with respect to position).

Note that the distribution is not of circular shape because the distance to the landmark is not used, but instead only the bearing to the landmark. This results in a radial distribution resembling magnetic field lines, i.e., the orientation of the particles is ordered, but not the position.

Incorporating negative information. The negative information gained in this experiment is not seeing but one landmark within the pan range. So in addition to the information that is derived from the landmark in front of the robot, knowledge about the position of the other three beacons and two goals can help update the belief because they are never detected. Incorporating this

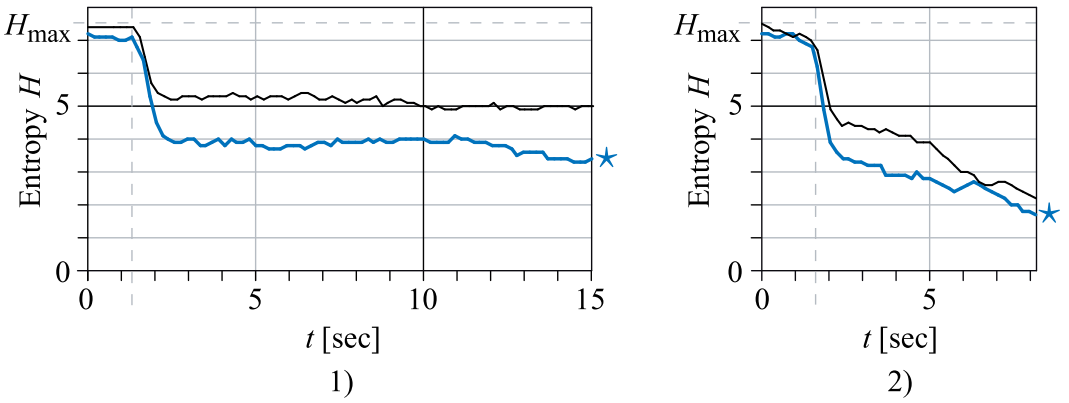


Figure 6.15: *Experiment E1:* Expected entropy of the belief in the localization task with ($\star$) and without (thin line) using negative information. 1) At first the robot does not see the landmark. As soon as the landmark comes into the robot's view (indicated by the dashed vertical line), the entropy drops. Using negative information, the quality of the localization is greatly improved and the entropy continues to decrease over time. 2) Additionally using field lines for localization enables the robot to localize even without negative information. Incorporating negative information, however, yields a higher rate of convergence and the entropy is still significantly lower.

information, the robot is able to localize quickly. On average, the robot is reasonably well localized after about 10 seconds with a pose error of less than $\Delta p = (25\,\text{cm}, 25\,\text{cm}, 20^\circ)$.

Expected Entropy

Fig. 6.15 shows the progression of the expected entropy during the experiment.

Not using negative information. The experiments starts with a uniform particle distribution, which results in maximum entropy. When the landmark comes into view, a decrease in entropy is observed. This information gain is due to the robot being able to now infer its relative orientation with respect to the landmark. Since there are no constraints on the robot's position, the entropy remains at a relatively high level. This is easily seen by separately calculating the entropy of the angle and position distributions. Note that, even though there is a small drop in entropy, the pose estimate is still highly uncertain.

Incorporating negative information. When using negative information, the entropy decreases even before the first sensor reading. The information

gain is much smaller than that caused by perceiving a landmark, but nevertheless noticeable. As soon as there is a percept, the negative information in combination with the knowledge of the robot's orientation result in a quick convergence towards the actual robot pose. This is remarkable since without using negative information, localization was not possible.

Using field lines for localization. The previous experiment was repeated using field lines for localization in addition to landmarks. Recall that when the robot detects a field line, the distance d from the robot to this line is used for localization.

Using field lines, the robot can localize quickly at the actual robot pose even without the use of negative information (Fig. 6.15, right), because the distance to lines constitutes additional information that further reduces the possible/probable positions on the field. Adding negative information, however, increases the rate of convergence and the overall level of entropy is reduced further. The decrease of entropy when incorporating negative information is not obscured by the use of lines for localization although field lines offer a greater information content than negative information. Information about field lines and negative information both contain information in the absence of a landmark percept. They are statistically independent, which can also be seen by the fact that the information gain brought about by negative information is still observed when line observations are integrated.

Localization Error

Fig. 6.16 shows the progression of the distribution's localization error over time for the above localization experiment calculated from the 2000 particle distribution. In the top half of the diagram, in addition to the localization error, the number of particles that were updated using negative information is also shown. In the bottom half, a record of the detected percepts is given.

Not using negative information. The experiment starts with a uniform particle distribution resulting in a large localization error. When the landmark comes into view, a small drop in localization error can be observed, but the error remains almost at the initial value for the remainder of the experiment.

Incorporating negative information. When using negative information, the particle distribution converges towards the actual robot pose, resulting in a low localization error. Looking at Fig. 6.11 and 6.12, you will notice that the range of the scan covers a large area of the field and that this area is bordering on a goal and a beacon. This affects how much the particle distribution

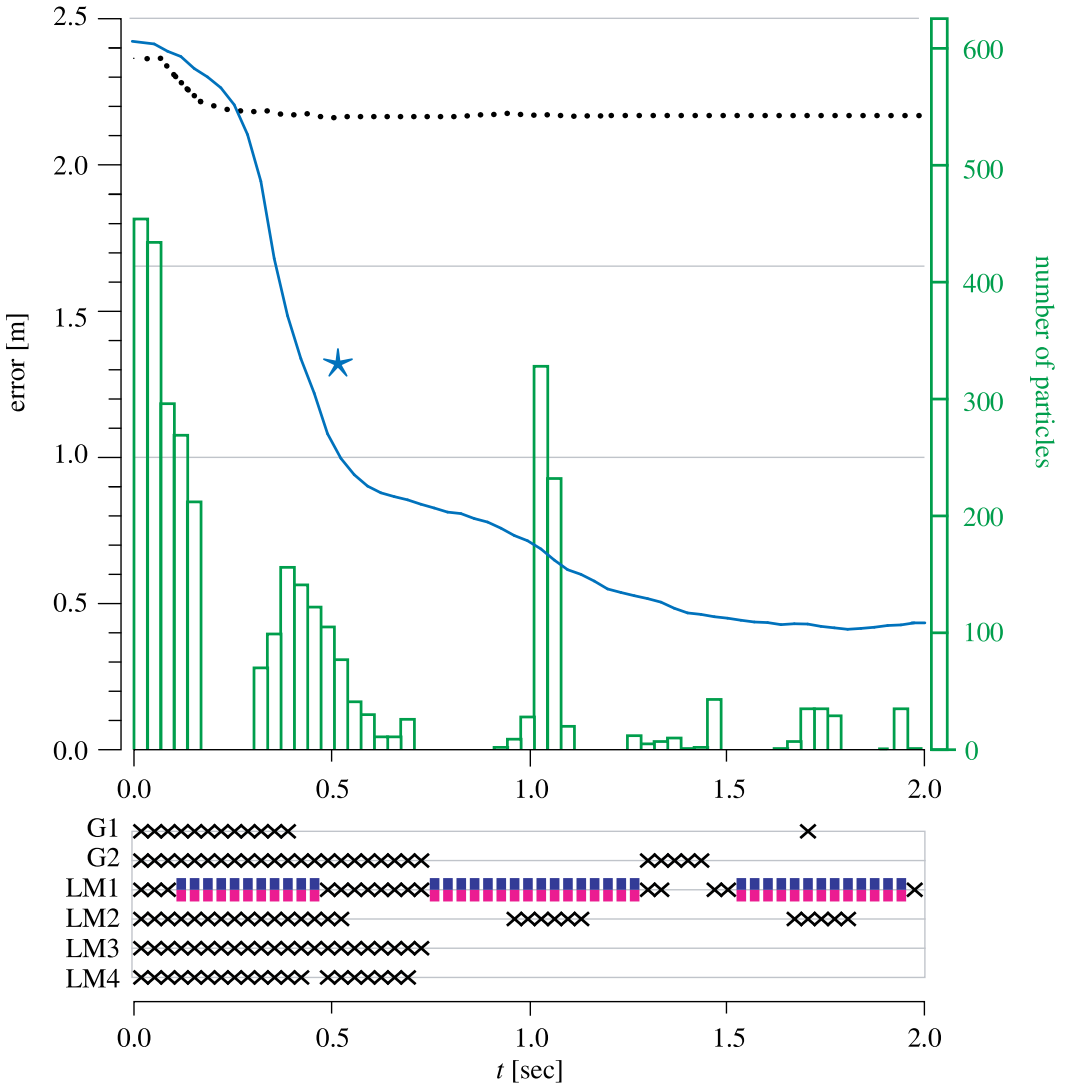


Figure 6.16: *Experiment E1:* Distance error of the localization. The dotted line shows the distance error without negative information, the solid blue line with negative information integrated. The green bars represent the number of particles, which are updated using negative information summed over all landmarks. *Bottom:* The diagram indicates if a landmark (LM) or goal (G) is seen (" | "), not seen ("−"), or expected but not seen ("x").

converges. In the case described here, almost a maximum of negative information is incorporated. The maximum is reached if the scan range cannot be increased any further without including the adjacent landmarks. In this case, the localization result is just as good as actually detecting the adjacent landmarks.

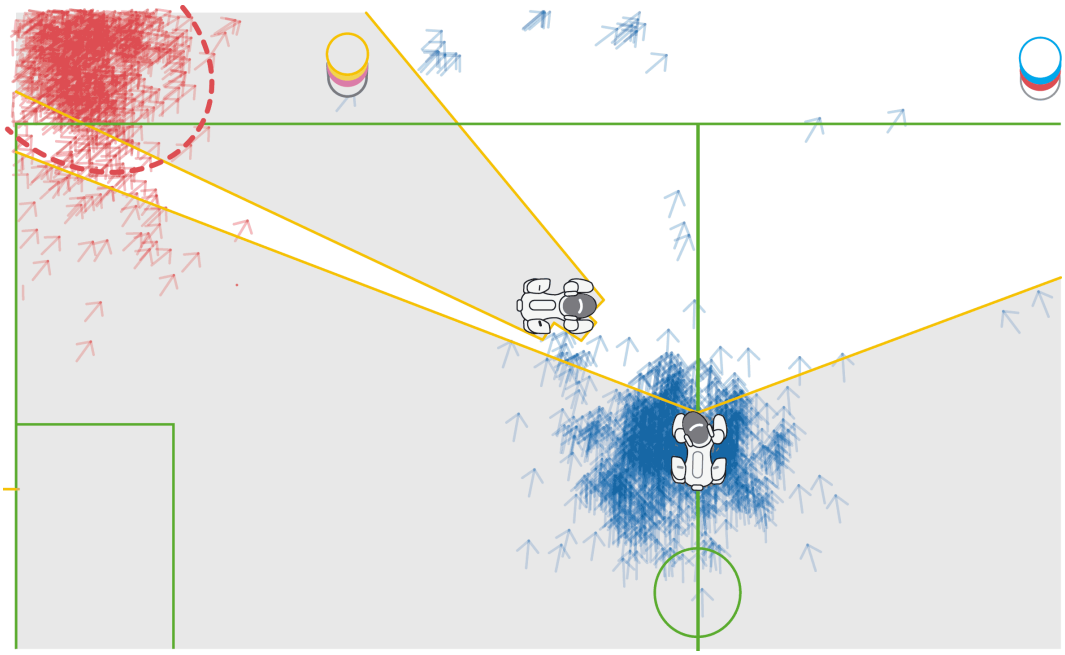


Figure 6.17: *Experiment E2:* In the situation shown, the landmark on the left is occluded by another robot. The blue particle distribution near the center circle shows the localization using negative information and modeling occlusions. When occlusions are not taken into account, false negatives will result in the particle distribution converging in a wrong area of the field (red particle distribution, marked by the dashed line).

Note that the number of particles that are updated using negative information decreases as the localization improves. Particles providing a good representation of the actual robot pose do not receive negative information.

6.4.2 E2 Two Landmarks, One Occluded

This experiment aims to illustrate the importance of properly modeling obstacles and occlusions. The robot is placed on the center line and it performs a scan as shown in Fig. 6.17. From where it is standing, two landmarks are within its sensing range. However, one of the landmarks is occluded by another robot.

As in the previous experiment, the standard approach is unable to localize; the resulting distribution is not shown here, but looks similar to the one found in the first experiment. In contrast, when negative information is incorporated taking into account occlusions, the robot is able to localize. Note that, although the illustration only shows part of the field, negative information

resulting from all landmarks on the field is used to update the belief.

The figure also shows the effect of not modeling occlusions correctly. Not modeling occlusions causes *false negatives*: the robot fails to see a landmark due to occlusions and wrongly assumes that it is absent from its sensing range. In these cases, the particle distribution converges to a wrong robot pose shown in the diagram. This pose is, of course, compatible with seeing a landmark at the right end of the scan and not seeing anything for the rest of the scan.

6.4.3 E3 Moving Robot

In this experiment, the robot walks on field following the ball. It illustrates the performance of the approach in an actual application. Fig. 6.18 shows the localization error and the particle distribution entropy in the first 3 seconds of a run.

In all four curves, landmark detection leads to an improvement in localization. When only percepts are used, distinct steps can be seen in the respective curves. As long as no new evidence is gathered, the level of uncertainty stays the same.

Incorporating negative information leads to a smoother, continuous decrease of uncertainty and to a better localization in the case of limited percepts. After some time, though, the robot has seen three landmarks and the quality of the localization reaches similar levels in both cases.

This illustrates one important aspect of negative information: it can help "fill in the blanks" in situations where there is little or incomplete sensor input. The quality of the resulting localization is limited by the best possible localization using all percepts potentially available at a given position. In other words: negative and positive evidence are two sides of a coin; a well localized robot cannot further improve its localization by negative evidence because there will be no negative evidence. This was already observed in the previous experiment, where the number of particles updated using negative information decreases as localization improves (Fig. 6.16).

6.4.4 E4 Kidnapped Robot

The kidnapped robot problem is a commonly used benchmark for the flexibility and robustness of localization algorithms [Gutmann et al., 1998; Gutmann and Fox, 2002]: a well-localized robot is displaced and the time for it to recover (re-localize) is measured.

Kidnapped robot experiments underline and confirm the already stated findings. The increased responsiveness of the localization when using negative information has a positive impact in the kidnapped robot benchmark.

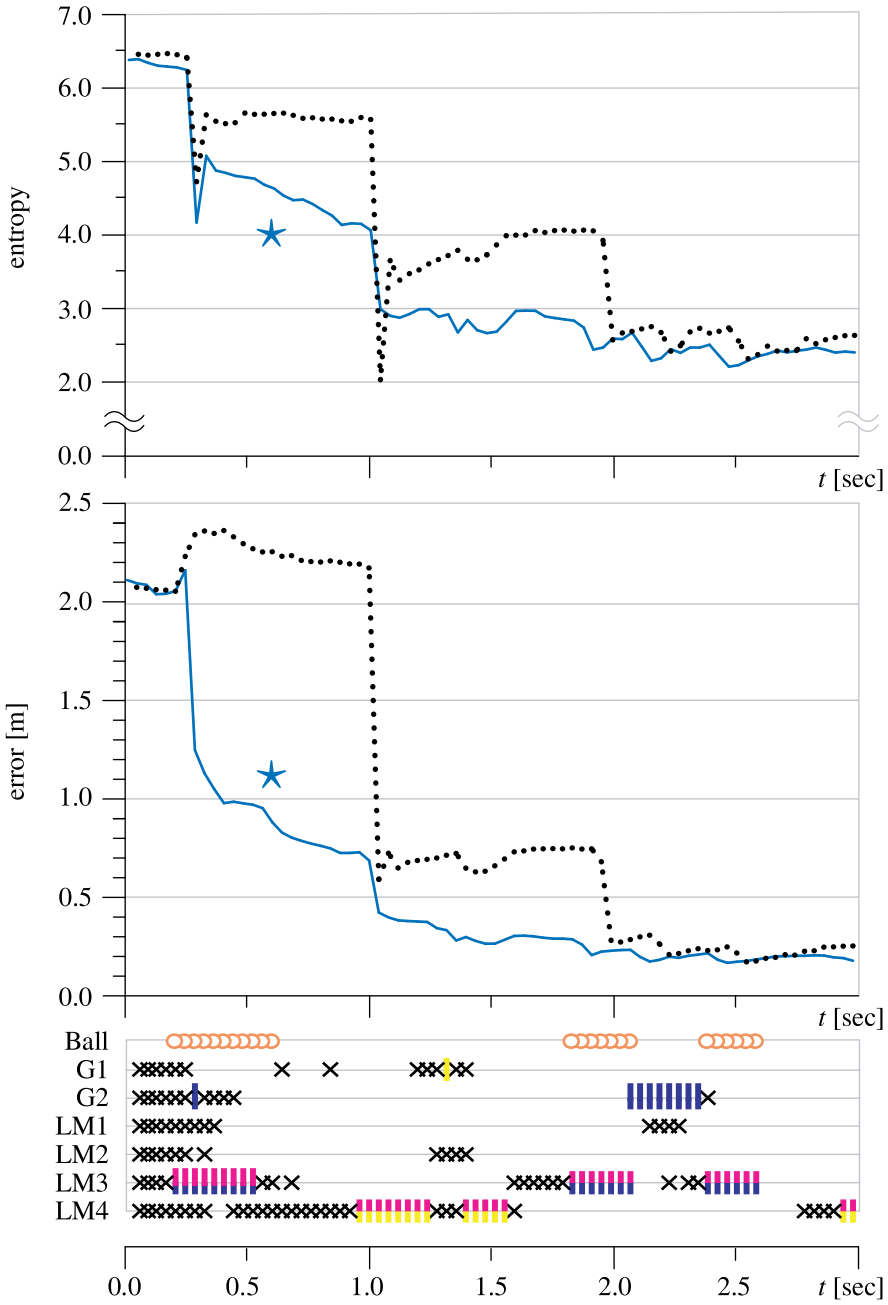


Figure 6.18: *Experiment E3:* Entropy of the particle distribution and localization error of a robot walking on the field chasing a ball. The solid, blue lines marked with the star represent the localization with negative information used, dotted lines without negative information. It shows that negative information is able to "fill in the blanks" before the next landmark is actually perceived.

One reason for this is the additional information that is used when the robot re-localizes, resulting in faster convergence of the particle distribution as shown above. But not only does the distribution converge more quickly, it also *diverges* more quickly in the absence of sensor readings that would confirm the robot's past position. The distribution therefore better resembles the actual situation of being lost and thus offers a much better starting point for subsequent re-localization. This is particularly helpful in RoboCup game situations where the robot often gets pushed by other robots. Unless collisions are explicitly modeled, these relatively small displacements can go unnoticed by the localization. Using negative information, the particle distribution diverges quickly to nearby positions, which often quite accurately models what has actually happened to the robot.

The ability to localize more quickly using negative information is highly beneficial in real world applications, where the robot is trying to actually perform a task rather than to localize perfectly. Such tasks often require the robot to focus its attention on objects other than landmarks and the sensing strategy may keep it from seeing as much of the world as it potentially could. Integrating negative evidence thus allows for more efficient sensing and improves overall robot performance.

6.5 Practical Considerations

Checking if a landmark lies within the viewing frustum is computationally expensive. The visibility check is performed in full 3D for a given robot pose and gaze direction. The presented results were generated off-line and in part in simulation to prove the concept. The code itself was not optimized for speed, which would be necessary for the approach to run on the Aibo. To speed up the visibility check, a look up table (LUT) can be used. The effective camera position and orientation is given by a by the 6 dimensional vector $(x, y, z, \text{pan}, \text{tilt}, \text{roll})$. All 6 dimensions are necessary to correctly describe the camera since the robot's body is tilted forward causing the head to also be tilted; a panning motion of the head thus causes the image to "roll," i.e., it is rotated along the vector pointing outward from the camera.

Neglecting the roll and combining robot orientation and head pan can reduce the dimensionality to 4 at a tolerable loss of accuracy. This can be achieved by only considering landmarks that would be in the center of the camera image where the effect of camera roll is small. With a spacing of 5 cm and 10° this results in $120 \times 80 \times 36 \times 18 \approx 6$ million entries. Storing only the binary information of whether or not the landmark is visible, this amounts to a LUT of approximately 800 kb per landmark. Taking into account field symmetry, only one LUT for beacons and one for goals needs to be used.

The computational burden can also be lessened by considering only the negative information of a limited number of landmarks during sensor update instead of all possible landmarks, similar to how only a subset of the detected lines percepts are used in lines-based localization.

6.6 Summary

Integrating negative information – the ascertained absence of an expected sensor reading – into Monte Carlo Localization yields improved localization performance as the available information is used more efficiently. Because sensors are more likely to overlook observable landmarks than hallucinate ones that are not visible, extra care has to be taken in designing the sensor model. To avoid false negatives, the model needs to take into account the sensor's sensing range and possible occlusions of landmarks.

Such a sensor model was implemented for a Sony Aibo robot and experimentally tested in the RoboCup environment. Real robot experiments show that using negative information, a robot is able to localize in positions where it otherwise would not have been able to localize. The robot senses a single landmark and, with the additional information of not seeing any other landmarks, can limit the area of where it believes to be. The entropy of the distribution is greatly reduced when negative information is incorporated and the rate of convergence towards the estimated position is increased. In accordance to entropy, the localization error is also decreased.

In actual application scenarios, complementary negative information is integrated over time and is thus able to fill in the gaps when few landmarks are seen. The information gain is bounded by the information that could be gathered if the robot was able to detect all landmarks.

This has interesting implications on perception in certain situations: it may make sense for the robot to direct its gaze towards areas where it does not actually see a landmark, but can rule out hypothesis. Looking in a direction where there are no landmarks may provide a larger information gain than looking at a landmark.

Chapter 7

Conclusion

The internal world model of a mobile robot needs to be accurate and comprehensive for it to perform a task in a dynamic environment. If the robot's world model is inaccurate, its internal belief differs from the state of the environment, resulting in actions that often seem erratic to the human observer. A robot running into a wall does so because it is unaware that it is standing right in front of the obstacle. Instead, it beliefs to be somewhere else in the environment and that it is safe to continue moving forward. The key challenge originates from the fact that the robot's environment is only partially observable, resulting in uncertainty associated with the robot's world model. Monte Carlo Localization (MCL) provides the robot with a particle filter representation of the world model that allows uncertainty to be modeled. The real-time constraint, in conjunction with limited processing power of the robot, imposes limitations on the MCL implementation, mainly limiting the number of particles that can be employed. MCL is generally good at modeling where the robot may be, but not very good at modeling where it is not, i.e., it is inefficient at modeling regions of low probability in state space. For these areas to be well represented, a large number of particles is needed. Thus, the random search of MCL becomes less efficient as the number of particles is decreased, resulting in reduced robustness with respect to coarsely modeled processes (motion and sensing) and unforeseen events (e.g., kidnapped robot). The effect for the robot in such situations is that its internal world model deviates from the true state of the environment, resulting in seemingly erratic robot behavior.

In this thesis, I explored ways to ensure the internal world model accurately describes the current state of the robot, and to thus make localization more robust and reactive even when the MCL particle count is limited.

I first presented a method to prevent the major source of such confusion based on an efficient system for collision avoidance. I then continued by taking a closer look at the Bayes filter used in localization and presented improvements to the motion update and to the sensor update of the filter: proprioceptive

information is used to better model robot locomotion and the notion of negative information in sensor modeling is introduced, allowing the robot to draw conclusions from the absence of feature detections.

Obstacle Modeling and Collision Avoidance

I presented a system for goal-directed obstacle avoidance. It is based on image processing determining free space in the camera image. This information is integrated into a two-layered, egocentric model of the vicinity of the robot, taking into account the robot's gaze direction. Maintaining a model is necessary, because the opening angle of the robot's camera is small and it can only see a small portion of its environment. Furthermore, the robot's focus of attention cannot be restricted to tracking obstacles when it is trying to achieve a more complex task in its environment. The obstacle model contains a radial representation of free space in conjunction with a buffer for recent measurements. It is accessed using analysis functions, which allow for reliable and robust collision avoidance. The system won the 2003 RoboCup obstacle avoidance challenge. It marks the basis for modeling visual occlusions and subsequently the use of negative information.

Proprioceptive Motion Model

I investigated how modeling the robot's motion can be improved by proprioception. This sense of the robot's own body is achieved by comparing control commands and actually performed movements. If the two deviate too much, the robot assumes the action to have failed. Such failure is commonly caused by collisions of the robot with static objects in the environment or with other robots. Collision detection can be used to trigger recovery actions, and more importantly, it can be used to more accurately model robot locomotion. Motion modeling for the Aibo is velocity-based and describes the robot's motion and the error associated with it. Using proprioceptive information, this error can be adjusted based on how well the robot is able to perform the desired action. If the motion is performed as intended (indicated by propriception), the locomotion error is small, e.g., when the robot is moving freely. If proprioception indicates a failure to execute the motion, the locomotion error increases, e.g., when the robot is running into an obstacle. Such differentiation was previously not possible: the motion model had to subsume everything from unhindered motion to collisions to running into walls. The world model benefits from the use of proprioception as the uncertainty introduced by locomotion is modeled more accurately. For periods of unhindered motion, the locomotion error remains small and the robot well localized. On the other hand, if a collision occurs, the motion and thus belief uncertainty quickly increase, reflecting

the unforeseen event. The resulting MCL particle distribution is more spread out, allowing for faster re-localization.

Proprioception can be achieved in the described way for any servo driven robot.

Negative Evidence Modeling

Lastly, I introduced the notion of negative information in modeling robot sensing. In contrast to dense sensor matching, feature-based localization utilizes features extracted from the sensor data. To my knowledge, all existing implementations only use the event of detecting a feature to update localization, hence feature extraction is commonly tuned in such a manner that few to no false positives occur. This is due to the fact that there are several reasons for the sensor failing to detect features, most importantly sensing imperfections, limited sensing range, and occlusions in the environment. I demonstrate how the latter two can be modeled explicitly to ascertain the absence of a feature in the robot's camera image (sensor imperfections are modeled probabilistically in the form of sensor noise). This ascertained non-detection event provides *negative information* and I show how it can be integrated into MCL. Negative evidence modeling uses the non-detection of a predicted observation to update the robot's belief. While many attempts at improving localization are based on adding features (e.g., field lines), integrating negative information helps make the best of the already available information. It serves as complementary information and can thus be used to improve localization, especially when landmarks are not within sight. This enables a robot pursuing a task to improve its localization, even by directing its gaze where there are no landmarks. Negative information fills in the sensory gaps between sensing landmarks. The maximum information gain from negative information is the same as that of actually sensing a landmark.

The presented approaches for modeling robot action and sensing constitute valuable enhancements of maintaining the robot's belief. The resulting belief is more reactive and is a better approximation of the robot's current state. Such a belief is well suited as a basis of probabilistic robot control.

7.1 Future Work

Active Saccadic Vision The presented work can be seen within the larger context of perception. The process of perception consists of state estimation (based on observations) and the act of sensing. Although not covered explicitly in this dissertation, active sensing was employed in actual applications, such

as robot soccer games. In games, multiple, disjunct localization hypothesis rarely occur. Instead, the belief is confined to a certain area, the size of which depends on the belief uncertainty. Basing active vision on the most likely robot pose estimate proved to greatly enhance overall robot performance. It enables the robot to quickly verify its current localization by directly gazing at expected landmarks, giving it more time to focus on tracking the ball, thus improving soccer performance. The internal model is used to guide the process of perception.

If solely the detection of landmarks is used to update the belief, the system will perform poorly when the belief is a poor approximation of the true position. The robot will then direct its gaze in a direction where it expects a landmark, but as the belief is inaccurate, it will fail to detect the landmark and it will gain no information from the sensing action. Using negative information about the expected but undetected landmark allows the robot to infer that the current localization is inaccurate, causing the MCL particle distribution to diverge. The robot's gaze direction will then change to account for the change in belief. As long as landmarks are not detected, negative information eliminates hypothesis that contradict the non-detection events. Eventually, the particle distribution converges towards the true pose.

This marks an important step towards a saccadic vision system, since failure to direct the gaze somewhere meaningful can be identified: the camera is pointed where, according to the current belief, a landmark is expected; if the camera fails to detect the landmark in this particular direction and no occluding obstacle is detected, the belief can be updated using negative information. If a system does not incorporate negative information, quickly pointing the gaze in a direction where no landmark is detected yields no information gain. The robot can end up looking in directions where there is nothing to detect and without its belief ever improving. Such systems therefore are limited to performing slow sweeping/panning motions with their cameras to make sure they detect landmarks even if they are not well localized. This is clearly less efficient than saccadic vision, where features can quickly be targeted and little time is spent looking at parts of the environment that hold no valuable information. Furthermore, sweeping the horizon for landmarks is only feasible in simple environments such as the RoboCup field. If features are found anywhere in 3D space, such sweeping motions become very time consuming. Saccadic vision utilizing negative information can therefore help the robot spend more of its attention on the actual task.

Calibration Proprioception To create a more accurate proprioceptive motion model, an external monitoring system needs to be employed. Machine learning approaches can then be used to derive a motion model that accu-

rately describes motion of the robot based on control and proprioceptive data. Ideally, training data is collected in life-like situations, e.g., during Robocup games or in game-like situations. Approaches to adapt the number of particles according to how well the belief is currently represented [Kwok et al., 2003] can then be employed to further reduce the computational burden and also to allow for probabilistic robot control.

Probabilistic Robot Control The presented improvements to MCL result in a robot belief that is more reactive and which better describes the situation the robot is in. This allows robot control to better take into account uncertainty, as well as the shape of the particle distribution. Methods of accessing and evaluating the world model should be investigated to help develop robust robot behaviors. For example, if uncertainty is modeled comprehensively, relocalization actions can be triggered based on how well the robot is localized. Furthermore, the concept of uncertainty with respect to a certain object or task may have interesting applications. A robot does not have to be well localized on the field to steer the ball into the opponent's goal. The information for it to successfully perform this task is its localization with respect to the goal. Previous approaches often resort to hybrid architectures to achieve such tasks, basing low level robot control directly on visual perception, e.g., by having the robot navigate towards the largest goal-colored blob in the camera image. While it is possible to design robust, efficient robot behavior in this way, the wealth of information contained in the robot's belief is left unused. Retrieving this information and putting it to use offers the chance of vastly enhancing robot behavior.

Bibliography

C. Agate, R. Wilkerson, and K. Sullivan. Utilizing negative information to track ground vehicles through move-stop-move cycles. In I. Kadar and E. Carapezza, editors, *Proceeding of Signal Processing, Sensor Fusion, and Target Recognition XIII (SPIE)*, volume 5429, pages 273–283, 2004.

R. Arkin. *Behavior-Based Robotics*. MIT Press, Cambridge, MA, USA, 1998.

S. Behnke. Local multiresolution path planning. In D. Polani, A. Bonarini, B. Browning, and K. Yoshida, editors, *7th International Workshop on RoboCup 2003 (Robot World Cup Soccer Games and Conferences)*, volume 3020 of *Lecture Notes in Artificial Intelligence*. Springer, 2004.

S. Behnke and R. Rojas. A hierarchy of reactive behaviors handles complexity. In M. Hannebauer, J. Wendler, and E. Pagello, editors, *Balancing Reactivity and Social Deliberation in Multi-Agent Systems*, volume 2103 of *Lecture Notes in Artificial Intelligence*, pages 125–126, 2000.

S. Behnke, J. Müller, and M. Schreiber. Toni: A soccer playing humanoid robot. In *Proceedings of The 9th RoboCup International Symposium 2005 (Robot World Cup Soccer Games and Conference)*, Lecture Notes in Artificial Intelligence. Springer, 2006. To appear.

R. Benosman and S. B. Kang. *Panoramic Vision: Sensors, Theory, and Applications*. Springer, 2001.

J. Borenstein and Y. Koren. The vector field histogram - fast obstacle avoidance for mobile robots. *The International Journal of Robotics Research*, 7 (3):278–288, 1991.

J. Borenstein and Y. Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(5):1179–1187, 1989.

V. Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. MIT Press, Cambridge, MA, USA, 1984.

R. A. Brooks. A robust layered control system for a mobile robot. In *IEEE Journal of Robotics and Automation*, number 1 in 2, pages 14–23, 1986.

J. Bruce and M. M. Veloso. Real-time randomized path planning for robot navigation. In *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2002.

J. Bruce, T. Balch, and M. Veloso. Fast and inexpensive color image segmentation for interactive robots. In *Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2061–2066. IEEE, 2000.

J. M. Buhmann, W. Burgard, A. B. Cremers, D. Fox, T. Hofmann, F. E. Schneider, J. Strikos, and S. Thrun. The mobile robot Rhino. *AI Magazine*, 16(2):31–38, 1995.

A. R. Cassandra, L. P. Kaelbling, and J. A. Kurien. Acting under uncertainty: Discrete bayesian models for mobile robot navigation. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 1996.

S. Chen, M. Siu, T. Vogelgesang, T. F. Yik, B. Hengst, S. B. Pham, and C. Sammut. The UNSW RoboCup 2001 sony legged robot league team. In A. B. 0002, S. Coradeschi, and S. Tadokoro, editors, *RoboCup 2001: Robot Soccer World Cup V*, volume 2377, pages 39–48, 2001. To appear.

J. E. Clark, J. G. Cham, S. A. Bailey, E. M. Froehlich, P. K. Nahata, R. J. Full, and M. R. Cutkosky. Biomimetic design and fabrication of a hexapedal running robot. In *Proceedings of the 2001 International Conference Robotics and Automation (ICRA)*, pages 3643–3649. IEEE, 2001.

J. Connell. *Minimalist Mobile Robotics: A Colony-style Architecture for an Artificial Creature.* Academic Press, 1990.

I. Dahm, D. Fisseler, M. Hebbel, and W. Nistico. Learning fast walking patterns with reliable odometry information for four-legged robots. In *Proceedings of the 15th International Symposium on Measurement and Control in Robotics 2005*, 2005. To appear.

B. D. Damas, P. U. Lima, and L. M. M. Custódio. A modified potential fields method for robot navigation applied to dribbling in robotic soccer. In G. A. Kaminka, P. U. Lima, and R. Rojas, editors, *RoboCup 2002: Robot Soccer World Cup VI*, volume 2752 of *Lecture Notes in Artificial Intelligence*, pages 65–77. Springer, 2003.

A. J. Davison and D. W. Murray. Mobile robot localisation using active vision. *Lecture Notes in Computer Science*, 1407, 1998.

J. R. del Solar and P. A. Vallejos. Motion detection and tracking for an aibo robot using camera motion compensation and kalman filtering. In D. Nardi, M. Riedmiller, C. Sammut, and J. Santos-Victor, editors, *8th International Workshop on RoboCup 2004 (Robot World Cup Soccer Games and Conferences)*, volume 3276 of *Lecture Notes in Artificial Intelligence*, pages 619–627. Springer, 2004.

F. Dellaert, W. Burgard, D. Fox, and S. Thrun. Using the condensation algorithm for robust, vision-based mobile robot localization. In *1999 Conference on Computer Vision and Pattern Recognition (CVPR '99)*, June 1999a.

F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte Carlo localization for mobile robots. In *Proceedings of the 1999 IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1322–1328. IEEE, 1999b.

U. Düffert and J. Hoffmann. Reliable and precise gait modeling for a quadruped robot. In I. Noda, A. Jacoff, A. Bredenfeld, and Y. Takahashi, editors, *9th International Workshop on RoboCup 2005 (Robot World Cup Soccer Games and Conferences)*, Lecture Notes in Artificial Intelligence. Springer, 2006. To appear.

D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

D. Fox, W. Burgard, and S. Thrun. Active markov localization for mobile robots. *Robotics and Autonomous Systems*, 1998a.

D. Fox, W. Burgard, S. Thrun, and A. B. Cremers. A hybrid collision avoidance method for mobile robots. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1238–1243. IEEE, 1998b.

D. Fox, W. Burgard, F. Dellart, and S. Thrun. Monte Carlo localization: Efficient position estimation for mobile robots. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence (AAAI)*, pages 343–349. The AAAI Press/The MIT Press, 1999a.

D. Fox, W. Burgard, and S. Thrun. Markov localization for reliable robot navigation and people detection. In *Proceedings of the Dagstuhl Seminar on*

Modelling and Planning for Sensor-Based Intelligent Robot Systems, 1999, 1999b.

M. Fujita and H. Kitano. Development of an autonomous quadruped robot for robot entertainment. *Autonomous Robots*, 5(1):7–18, 1998.

A. Gloye, C. Göktekin, A. Egorova, O. Tenchio, and R. Rojas. Learning to drive and simulate autonomous mobile robots. In D. Nardi, M. Riedmiller, C. Sammut, and J. Santos-Victor, editors, *8th International Workshop on RoboCup 2004 (Robot World Cup Soccer Games and Conferences)*, volume 3276 of *Lecture Notes in Artificial Intelligence*. Springer, 2005.

K. Gunnarsson, F. Wiesel, and R. Rojas. The color and the shape: Automatic on-line color calibration for autonomous robots. In I. Noda, A. Jacoff, A. Bredenfeld, and Y. Takahashi, editors, *9th International Workshop on RoboCup 2005 (Robot World Cup Soccer Games and Conferences)*, Lecture Notes in Artificial Intelligence. Springer, 2006. To appear.

J.-S. Gutmann and D. Fox. An experimental comparison of localization methods continued. In *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2002.

J.-S. Gutmann, W. Burgard, D. Fox, and K. Konolige. An experimental comparison of localization methods. In *Proceedings of the 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 1998.

R. Hanek, T. Schmitt, S. Buck, and M. Beetz. Fast image-based object localization in natural scenes. In *Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2002.

R. Hanek, T. Schmitt, S. Buck, and M. Beetz. Towards RoboCup without color labeling. In G. A. Kaminka, P. U. Lima, and R. Rojas, editors, *6th International Workshop on RoboCup 2002 (Robot World Cup Soccer Games and Conference)*, volume 2752 of *Lecture Notes in Artificial Intelligence*, pages 179–194. Springer, 2003.

J. Hoffmann. Proprioception based motion modeling for Monte Carlo localization. In G. Lakemeyer, E. Sklar, D. Sorrenti, and T. Takahashi, editors, *10th International Workshop on RoboCup 2006 (Robot World Cup Soccer Games and Conference)*, Lecture Notes in Artificial Intelligence. Springer, 2007. To appear.

J. Hoffmann and D. Göhring. Sensor-actuator-comparison as a basis for collision detection for a quadruped robot. In D. Nardi, M. Riedmiller, C. Sammut, and J. Santos-Victor, editors, *8th International Workshop on RoboCup 2004 (Robot World Cup Soccer Games and Conferences)*, volume 3276 of *Lecture Notes in Artificial Intelligence*, pages 150–159. Springer, 2005.

J. Hoffmann, M. Jüngel, and M. Lötzsch. A vision based system for goal-directed obstacle avoidance. In D. Nardi, M. Riedmiller, C. Sammut, and J. Santos-Victor, editors, *8th International Workshop on RoboCup 2004 (Robot World Cup Soccer Games and Conferences)*, volume 3276 of *Lecture Notes in Artificial Intelligence*, pages 418–425. Springer, 2005.

J. Hoffmann, M. Spranger, D. Göhring, and M. Jüngel. Making use of what you don't see: Negative information in markov localization. In *Proceedings of the 2005 IEEE/RSJ International Conference of Intelligent Robots and Systems (IROS)*. IEEE, 2006a. To appear.

J. Hoffmann, M. Spranger, D. Göhring, and M. Jüngel. Exploiting the unexpected: Negative evidence modeling and proprioceptive motion modeling for improved markov localization. In I. Noda, A. Jacoff, A. Bredenfeld, and Y. Takahashi, editors, *9th International Workshop on RoboCup 2005 (Robot World Cup Soccer Games and Conference)*, Lecture Notes in Artificial Intelligence. Springer, 2006b. To appear.

J. Hoffmann, M. Spranger, D. Göhring, M. Jüngel, and H.-D. Burkhard. Further studies on the use of negative information in mobile robot localization. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2006c. To appear.

G. Hornby, S. Takamura, J. Yokono, O. Hanagata, T. Yamamoto, and M. Fujita. Evolving robust gaits with Aibo. In *Proceedings of the 2000 International Conference on Robotics and Automation (ICRA)*, pages 3040–3045. IEEE, 2000.

I. Horswill. Visual collision avoidance by segmentation. In *Proceedings of the 1994 IEEE/RSJ International Conference of Intelligent Robots and Systems (IROS)*, pages 902–909. IEEE, 1994.

I. Horswill. Polly: A vision-based artificial agent. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI-93)*, pages 824–829. The AAAI Press/The MIT Press, 1993.

K. Iagnemma, C. Brooks, and S. Dubowsky. Visual, tactile, and vibration-based terrain analysis for planetary rovers. In *Proceedings of the 2004 IEEE Aerospace Conference*, volume 2, pages 841– 848. IEEE, 2004.

M. Jüngel, J. Hoffmann, and M. Lötzsch. A real-time auto-adjusting vision system for robotic soccer. In D. Polani, A. Bonarini, B. Browning, and K. Yoshida, editors, *7th International Workshop on RoboCup 2003 (Robot World Cup Soccer Games and Conferences)*, volume 3020 of *Lecture Notes in Artificial Intelligence*, pages 214–225. Springer, 2004.

R. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME - Journal of Basic Engineering*, 82:35–45, 1960.

O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1), 1986.

H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. RoboCup: The robot world cup initiative. In W. L. Johnson and B. Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 340–347, New York, 5–8, 1997a. ACM Press. ISBN 0-89791-877-0.

H. Kitano, Y. Kuniyoshi, I. Noda, M. Asada, H. Matsubara, and E. Osawa. Robocup: A challenge for AI. *Artificial Intelligence Magazine*, 18(1):73–85, 1997b.

W. Koch. On negative information in tracking and sensor data fusion. In *Proceedings of the Seventh International Conference on Information Fusion*, pages 91–98, 2004.

C. Kwok and D. Fox. Map-based multiple model tracking of a moving object. In D. Nardi, M. Riedmiller, C. Sammut, and J. Santos-Victor, editors, *8th International Workshop on RoboCup 2004 (Robot World Cup Soccer Games and Conferences)*, volume 3276 of *Lecture Notes in Artificial Intelligence*, pages 18–33. Springer, 2005.

C. Kwok, D. Fox, and M. Meila. Adaptive real-time particle filters for robot localization. In *Proceedings of the 2003 IEEE International Conference on Robotics & Automation*, 2003.

C. Kwok, D. Fox, and M. Meila. Real-time particle filters. *Proceedings of the IEEE, Special Issue on Sequential State Estimation*, 92(2), 2004.

A. Lankenau, T. Röfer, and B. Krieg-Brückner. Self-localization in large-scale environments for the Bremen autonomous wheelchair. In C. Freksa, W. Brauer, C. Habel, and K. F. Wender, editors, *Spatial Cognition III, Routes and Navigation, Human Memory and Learning, Spatial Representation and Spatial Learning*, volume 2685 of *Lecture Notes in Artificial Intelligence*. Springer, 2002.

J. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, USA, 1991.

T. Laue and T. Röfer. A behavior architecture for autonomous mobile robots based on potential fields. In D. Nardi, M. Riedmiller, C. Sammut, and J. Santos-Victor, editors, *8th International Workshop on RoboCup 2004 (Robot World Cup Soccer Games and Conferences)*, volume 3276 of *Lecture Notes in Artificial Intelligence*, pages 122–133. Springer, 2005.

T. Laue, K. Spiess, and T. Röfer. SimRobot - A general physical robot simulator and its application in robocup. In I. Noda, A. Jacoff, A. Bredenfeld, and Y. Takahashi, editors, *9th International Workshop on RoboCup 2005 (Robot World Cup Soccer Games and Conference)*, Lecture Notes in Artificial Intelligence. Springer, 2006. To appear.

S. Lenser and M. Veloso. Visual sonar: Fast obstacle avoidance using monocular vision. In *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2003.

S. Lenser and M. M. Veloso. Sensor resetting localization for poorly modelled mobile robots. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation (ICRA 2000)*, pages 1225–1232. IEEE, 2000.

S. Lenser, J. Bruce, and M. Veloso. CMPack: A complete software system for autonomous legged soccer robots. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 204–211. ACM Press, 2001. ISBN 1-58113-326-X.

L. Lorigo, R. Brooks, and W. Grimson. Visually guided obstacle avoidance in unstructured environments. In *Proceedings of the 1997 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 373–379. IEEE, 1997.

M. Lötzsch, J. Bach, H.-D. Burkhard, and M. Jüngel. Designing agent behavior with the extensible agent behavior specification language XABSL. In D. Polani, B. Browning, A. Bonarini, and K. Yoshida, editors, *RoboCup 2003: Robot Soccer World Cup VII*, volume 3020 of *Lecture Notes in Artificial Intelligence*. Springer, 2004.

E. Menegatti, A. Pretto, and E. Pagello. A new omnidirectional vision sensor for Monte-Carlo localization. In D. Nardi, M. Riedmiller, C. Sammut, and J. Santos-Victor, editors, *8th International Workshop on RoboCup 2004 (Robot World Cup Soccer Games and Conferences)*, volume 3276 of *Lecture Notes in Artificial Intelligence*, pages 97–109. Springer, 2005.

N. Molton and M. Brady. Practical structure and motion from stereo when motion is unconstrained. *International Journal of Computer Vision*, 39(1): 5–23, 2000.

M. Montemerlo and S. Thrun. Simultaneous localization and mapping with unknown data association using FastSLAM. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1985–1991. IEEE, 2003.

W. Nisticò and T. Röfer. Improving percept reliability in the sony four-legged robot league. In I. Noda, A. Jacoff, A. Bredenfeld, and Y. Takahashi, editors, *9th International Workshop on RoboCup 2005 (Robot World Cup Soccer Games and Conference)*, Lecture Notes in Artificial Intelligence. Springer, 2006. To appear.

C. L. P. Dario, E. Guglielmelli. Humanoids and personal robots: design and experiments. *Journal of Robotic Systems*, 18(2), 2001.

M. J. Quinlan, C. L. Murch, R. H. Middleton, and S. K. Chalup. Traction monitoring for collision detection with legged robots. In D. Polani, A. Bonarini, B. Browning, and K. Yoshida, editors, *7th International Workshop on RoboCup 2003 (Robot World Cup Soccer Games and Conference)*, volume 3020 of *Lecture Notes in Artificial Intelligence*, pages 374–384. Springer, 2004.

R. Rojas, S. Behnke, A. Liers, and L. Knipping. FU-Fighters 2001 (global vision). In A. B. 0002, S. Coradeschi, and S. Tadokoro, editors, *RoboCup 2001: Robot Soccer World Cup V*, volume 2377 of *Lecture Notes in Artificial Intelligence*, pages 204–213. Springer, 2002.

N. Roy, W. Burgard, D. Fox, and S. Thrun. Coastal navigation – mobile robot navigation with uncertainty in dynamic environments. In *Proc. of the 1999 IEEE International Conference on Robotics and Automation (ICRA)*, pages 35–40. IEEE, 1999.

T. Röfer. Evolutionary gait-optimization using a fitness function based on proprioception. In D. Nardi, M. Riedmiller, C. Sammut, and J. Santos-Victor, editors, *8th International Workshop on RoboCup 2004 (Robot World Cup Soccer Games and Conferences)*, volume 3276 of *Lecture Notes in Artificial Intelligence*, pages 310–322. Springer, 2005.

T. Röfer and M. Jüngel. Fast and robust edge-based localization in the sony four-legged robot league. In D. Polani, A. Bonarini, B. Browning, and K. Yoshida, editors, *7th International Workshop on RoboCup 2003 (Robot*

World Cup Soccer Games and Conferences), volume 3020 of *Lecture Notes in Artificial Intelligence*, pages 262–273. Springer, 2004.

T. Röfer and M. Jüngel. Vision-based fast and reactive Monte-Carlo localization. In D. Polani, A. Bonarini, B. Browning, and K. Yoshida, editors, *Proceedings of the 2003 IEEE International Conference on Robotics and Automation (ICRA)*, pages 856–861. IEEE, 2003.

T. Röfer, R. Brunn, I. Dahm, M. Hebbel, J. Hoffmann, M. Jüngel, T. Laue, M. Lötzsch, W. Nistico, and M. Spranger. GermanTeam 2004: The German national RoboCup team. In D. Nardi, M. Riedmiller, C. Sammut, and J. Santos-Victor, editors, *RoboCup 2004: Robot Soccer World Cup VIII*, volume 3276 of *Lecture Notes in Artificial Intelligence*. Springer, 2005. Full team report can be downloaded at http://www.germanteam.org/.

T. Röfer, T. Laue, and D. Thomas. Particle-filter-based self-localization using landmarks and directed lines. In I. Noda, A. Jacoff, A. Bredenfeld, and Y. Takahashi, editors, *9th International Workshop on RoboCup 2005 (Robot World Cup Soccer Games and Conference)*, Lecture Notes in Artificial Intelligence. Springer, 2006. To appear.

O. Sachs. *The Man Who Mistook His Wife For A Hat And Other Clinical Tales*, chapter 3, pages 43–54. Touchstone, New York, NY, USA, 1985.

D. Schulz and D. Fox. Bayesian color estimation for adaptive vision-based robot localization. In *Proceedings of the 2004 IEEE/RSJ International Conference of Intelligent Robots and Systems (IROS)*. IEEE, 2005.

R. Schuster, N. Ansari, and A. Bani-Hashemi. Steering a robot with vanishing points. *IEEE Trans. Robotics and Automation*, 9(4):491–498, Aug. 1993.

J. T. Schwartz, M. Sharir, and J. E. Hopcroft, editors. *Planning, geometry, and complexity of robot motion.* Ablex Publishing Corp., Norwood, NJ, USA, 1986.

D. Sekimori, T. Usui, Y. Masutani, and F. Miyazaki. High-speed obstacle avoidance and self-localization for mobile robots based on omni-directional imaging of floor region. In A. B. 0002, S. Coradeschi, and S. Tadokoro, editors, *RoboCup 2001: Robot Soccer World Cup V*, volume 2377 of *Lecture Notes in Artificial Intelligence*, pages 204–213. Springer, 2002.

R. Simmons. The curvature velocity method for local obstacle avoidance. In *Proceedings of the 1996 International Conference on Robotics and Automation (ICRA)*, pages 3375–3382. IEEE, 1996.

M. Simon, F. Wiesel, A. Egorova, A. Gloye, and R. Rojas. Plug & play: Fast automatic geometry and color calibration for tracking mobile robots. In D. Nardi, M. Riedmiller, C. Sammut, and J. Santos-Victor, editors, *8th International Workshop on RoboCup 2004 (Robot World Cup Soccer Games and Conferences)*, volume 3276 of *Lecture Notes in Artificial Intelligence*, pages 394–401. Springer, 2005.

L. Steels. Towards a theory of emergent functionality. In J.-A. Meyer and R.Wilson, editors, *From animals to animats: Proceedings of the first international conference on simulation of adaptive behavior*, pages 451–461, Cambridge, MA, USA, 1991. MIT Press. ISBN 0-262-63138-5.

S. Särkkä, T. Tamminen, A. Vehtari, and J. Lampinen. Probabilistic methods in multiple target tracking, research report b36. Technical report, Laboratory of Computational Engineering, Helsinki University of Technology, 2004.

S. Thrun, D. Fox, and W. Burgard. Monte Carlo localization with mixture proposal distribution. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 859–865, 2000.

S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, Cambridge, MA, USA, 2005.

D. Vail and M. Veloso. Learning from accelerometer data on a legged robot. In *Proceedings of the 5th IFAC Symposium on Intelligent Autonomous Vehicles (IAV-2004)*, 2004.

F. von Hundelshausen, M. Schreiber, and R. Rojas. A constructive feature detection approach for robotic vision. In D. Nardi, M. Riedmiller, C. Sammut, and J. Santos-Victor, editors, *8th International Workshop on RoboCup 2004 (Robot World Cup Soccer Games and Conferences)*, volume 3276 of *Lecture Notes in Artificial Intelligence*, pages 72–83. Springer, 2005.

T. Weigel, A. Kleiner, F. Diesch, M. Dietl, J.-S. Gutmann, B. Nebel, P. Stiegeler, and B. Szerbakowski. CS Freiburg 2001. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup 2001: Robot Soccer World Cup V*, Lecture Notes in Artificial Intelligence, pages 26–38. Springer, 2002.

J. Wendler, S. Brüggert, H.-D. Burkhard, and H. Myritz. Fault-tolerant self localization by case-based reasoning. In P. Stone, T. Balch, and G. Kraetzschmar, editors, *RoboCup 2000: Robot Soccer World Cup IV*, volume 2019, pages 259–268. Springer, 2001.

K. Yoshida, H. Hamano, and T. Watanabe. Slip-based traction control of a planetary rover. In *Experimental Robotics VIII*, Advanced Robotics Series. Springer, 2002.

Acknowledgements

I would like to thank my supervisor Hans-Dieter Burkhard for giving me the opportunity to work in the field of robotics and for believing in me. He supported me, as I sought to find my own way, which admittedly was not always easy. He provided a friendly work environment and supported my research efforts through many fruitful discussions. He also allowed me to take responsibility for the team. I would also like to acknowledge my co-supervisors, Raul Rojas and Uwe Schwiegelshohn, for their time and support.

I wish to thank the members of Aibo Team Humboldt, Benjamin Altmeyer, Daniel Göhring, Uwe Düffert, Matthias Jüngel, Martin Lötzsch, and Michael Spranger. They shared in many discussions, demonstrated excellent work, especially provided a solid framework for mobile robotics, and let me be part of a very successful team. My gratitude extends also to the members of the GermanTeam, who made this project exciting and successful on a scientific, competitive, and personal level: Ingo Dahm, Matthias Hebbel, and Walter Nistico of Dortmund University, Thomas Röfer and Tim Laue of Bremen University, and Max Riesler of Technical University Darmstadt. Working with them on the GermanTeam and sharing many crunch time moments at the various competitions is a memorable, treasured experience of mine.

I would like to thank Freek Stulp, who was a great help, providing creative input and constructive criticism regarding publications and this thesis. He drew my attention to the Sherlock Holmes text "Silver Blaze," used to illustrate the notion of negative information. The collaboration within the DFG priority project also provided me with the opportunity to meet and discuss ideas with Felix von Hundelshausen, Thorsten Schmitt, Ansgar Bredenfeld, and Keyan Zahedi, for which I am grateful.

I would also like to thank Dieter Fox for inviting me to visit his group in Seattle, an experience I would not want to miss. I would also like to acknowledge Cody Kwok, Jonathan Ko, and Seth Bridges, who made me feel welcome and also supported me sharing in discussions and providing suggestions for my work and its presentation. The same is true for Mohan Sridharan and Vladimir Estivill-Castro, who I had the pleasure of getting to know while participating in RoboCup. A special "thank you" belongs to Vlad for making it possible for

me to visit his lab in Brisbane.

I would like to thank all other members of our group, Olga Schiemangk, Renate Zirkelbach, Kay Schröter, Joscha Bach, and the RoboCup simulation team, Ralf Berger, Daniel Hein, and Michael Gollin, for their support and care.

At last, but certainly not least, I would like to extend my gratitude to Carolin, my parents Annette and Wolf, to Marlene, and to my friends for their help, patience, and understanding.

Funding The project was funded by the German Research Foundation (DFG), Priority Project 1125 "Cooperative Teams of Mobile Robots in Dynamic Environments."

Selbständigkeitserklärung

Hiermit erkläre ich, dass

1. ich die vorliegende Dissertationsschrift “Reactive Probabilistic Belief Modeling for Mobile Robots” selbständig und ohne unerlaubte Hilfe angefertigt habe,

2. ich mich nicht anderswärts um einen Doktorgrad beworben habe oder einen solchen besitze,

3. mir die Promotionsordnung der Mathematisch-Naturwissenschaftlichen Fakultät II der Humboldt-Universität zu Berlin bekannt ist.

Jan Hoffmann
Berlin, den 6.7.2007